FIREWALL

AVEC

http://www.netfilter.org/

(Site Officiel de netfilter)

Ce document à été réalisé essentiellement à partir d'autres documents trouvés sur Internet dont principalement :

http://olivieraj.free.fr/fr/linux/information/firewall/

http//www.netfilter.org/

et aussi à partir d'articles parrus dans :

Linux magazine : http://www.linuxmag-france.org/
Login : http://www.login.fr/
MISC : http://www.miscmag.com/

Il n'est toujours pas terminé ! …

NETFILTER / IPTABLES

Sommaire :

I. Vue générale de Netfilter

Netfilter est le firewall de Linux 2.4 et supérieur.

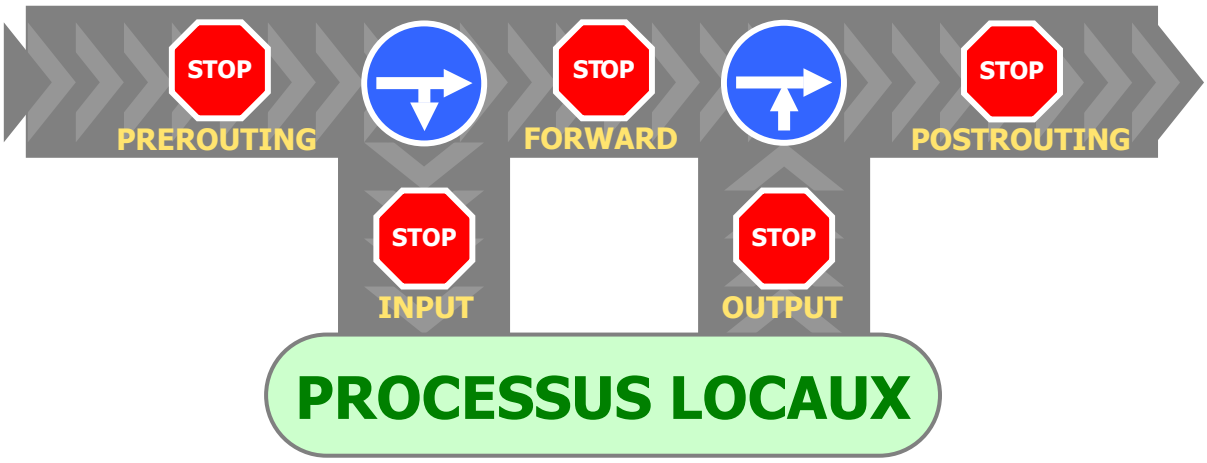
Netfilter est le nom d'une partie du noyau Linux qui est destinée à assurer la surveillance de tous les transferts de données réseaux. Sa tache est de faire du "**Network Packet Filtering**", c'est à dire du "Filtrage de Paquets Réseaux". On peut considérer qu'il se place entre la couche réseau du noyau Linux, et la couche applicative (c'est en réalité un peu plus compliqué que ça !). Netfilter supporte actuellement les protocoles IPv4, IPv6, DECnet, et ARP, et en partie IPX via des patchs expérimentaux. Nous ne nous intéresserons ici qu'à IPv4.

Comme Netfilter est un élément implanté profondément dans le noyau Linux (le "*kernel space*", ou "l'espace du noyau" en français), on ne peut pas le configurer aussi simplement qu'avec un jolie interface graphique doté de pleins de couleurs. On ne peut pas non plus le paramétrer directement via un fichier de configuration du "/etc/". L'unique moyen que nous ayons de dialoguer avec lui est un programme appelé "**iptables**", que l'on lance dans le "*user space*" ("espace utilisateur" en français) **avec les droits root**. Il existe cependant un certain nombre d'autres applications, graphiques ou non, qui servent d'interface à "iptables". (shorewall par exemple)

Pour ceux qui veulent en savoir un peu plus sur le fonctionnement de Netfilter, je les invite à consulter la documentation du développeur de Netfilter (traduite en français >> http://www.netfilter.org/).

Netfilter peut intervenir en 5 endroits du système de gestion de la pile IP. Pour chacune de ces étapes (que l'on appelle des "**hook**", pour "crochet" en français), Netfilter peut :

- Imposer au noyau de ***supprimer le paquet*** ("**drop**" en anglais). Auquel cas, il est jeté aux oubliettes, et c'est comme si le paquet n'avait jamais existé.
- Indiquer au noyau que ***le paquet est accepté***. Dans ce cas là, le noyau peut continuer à travailler dessus.
- ***Modifier le paquet***, puis le rendre au noyau.



Il est assez facile de comprendre ce qu'est un "**hook**". C'est l'équivalent dans la couche IP d'une interruption (matérielle ou logicielle) : Lorsque qu'un événement arrive, le système d'exploitation arrête les tâches en cours, et exécute un morceau de code particulier. Une fois que ce code est fini, le système continue son travail comme si rien ne s'était passé.

Pour chacun de ces "**hook**", **Netfilter associe une chaîne**. Mais qu'est-ce qu'une chaîne ? **Une chaîne est un ensemble de règles** (du type "si quelque chose alors je fais ceci") concernant les paquets IP : leur origine, leur destination, leur taille, etc... En fonction des différentes règles de la chaîne, **Netfilter** pourra décider quoi fait du paquet IP : Le laisser passer, le supprimer ou le modifier.

Faisons une analogie avec la route : Les paquets IP sont des voitures, qui circulent sur des routes. Si Netfilter n'est pas configuré spécialement, les voitures se déplacerons à travers les différentes routes de la pile IP, emprunteront les ronds-points de routage, puis finirons par sortir de la route IP. Par contre, si Netfilter est activé, il y aura des barrages de police en certains endroits du trajet des paquets (les fameux "**hook**"). En fonction des règles contenus dans ces "**hook**" (les fameuses chaînes, c'est à dire les signes "**STOP**"), le barrage décidera ou non de laisser passer le véhicule IP, voir de le modifier... D'ailleurs, c'est un peu ce qui se passe sur nos routes à nous. Il y a parfois des contrôles de police ou de gendarmerie : En fonctions de certaines règles (contrôle des papiers, de l'alcoolémie, de l'état du véhicule, du délit de "sale gueule", ...) l'officier pourra ou non décider de laisser passer le véhicule...

Voyons maintenant les 5 "hook" et les 5 chaînes qui y sont associées :

Hook	*Chaîne*	*Description*
NF_IP_PRE_ROUTING	**PREROUTING**	A ce stade, le paquet est "brut de forme", c'est à dire qu'il n'a subit aucune modification par rapport à ce que l'interface réseau a reçu. On reparlera de ce hook un peu plus tard, donc ce n'est pas la peine de s'y intéresser tout de suite.
NF_IP_LOCAL_IN	**INPUT**	Ce "hook" est très intéressant, car à ce stade, le paquet est prêt à être envoyé aux couches applicatives, c'est à dire aux serveurs et aux clients qui tournent sur la machine. C'est un des points principaux sur lequel nous allons travailler.
NF_IP_FORWARD	**FORWARD**	"Forward" ("faire suivre" en français) est assez particulier. Ce "hook" voit passer des paquets IP qui vont transiter d'une interface réseau à une autre, sans passer par la couche applicative. Pourquoi diantre faire suivre un paquet entre 2 interfaces réseaux, comme par exemple entre "eth0" et "ppp0" ? En fait, c'est afin de permettre à Linux de se transformer en passerelle : Vous vous souvenez, c'est l'histoire du garde-barrière que nous avons vu dans le 1er chapitre. Comme pour "NF_IP_PRE_ROUTING", nous n'en reparlerons que plus tard.
NF_IP_LOCAL_OUT	**OUPUT**	Ce "hook" est l'équivalent du "NF_IP_LOCAL_IN", sauf qu'il est exécuté après que les couches applicatives aient traités, ou générés, un paquet IP. Tout comme "NF_IP_LOCAL_IN", c'est un point que nous allons voir en détail.
NF_IP_POSTROUTING	**POSTROUTING**	C'est l'équivalent du "NF_IP_PRE_ROUTING" pour les paquets IP sortants de la couche IP. A ce stade, les paquets sont prêt à être envoyés sur l'interface réseau. Comme pour "NF_IP_PRE_ROUTING" et "NF_IP_FORWARD", nous en parlerons plus tard.

Le graphique de la page précédente nous montre l'implantation des différents **hooks**. Ce qui nous intéresse le plus est la possibilité de contrôler les paquets arrivant et sortant des différents logiciels de notre machine. C'est donc au niveau des "**hooks**" "**NF_IP_LOCAL_IN**" et "**NF_IP_LOCAL_OUT**" que nous allons nous pencher plus précisément, via la notion de filtrage ("**filter**" ou "filtering" en anglais).

II. Les tables

Une table permet de définir un comportement précis de **Netfilter**. Une **table** est en fait un **ensemble de chaînes**, elles-mêmes **composées de règles** . Une table va donc nous permettre de manipuler **Netfilter**, afin de lui faire réaliser des choses intéressantes.

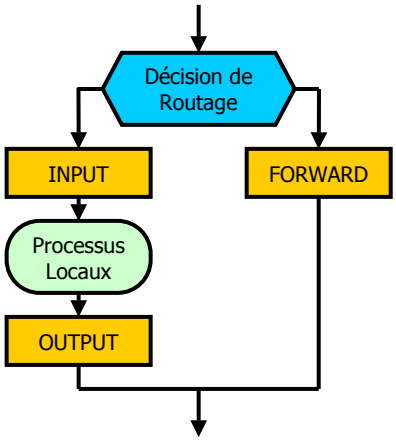
Mais comment manipuler ces tables ? … avec le fameux programme "**iptables**"

Il existe pour l'instant ***3 tables*** (**FILTER**, **NAT** et **MANGLE**), d'autres pouvant être rajoutées à l'avenir. Nous allons nous intéresser principalement à la table "**Filter**", puis dans un 2nd temps à la table "**NAT**". Nous ne ferons qu'évoquer la table "**Mangle**", car elle n'a pas beaucoup d'intérêt ici.

A. La table FILTER

Comme son nom l'indique, cette table **sert à filtrer les paquets** réseaux. C'est à dire que nous allons pouvoir trier les paquets qui passent à travers le réseau, et supprimer ceux qui ne nous intéressent pas, ou que nous trouvons dangereux.

Pour cela, la table "**Filter**" n'utilise que 3 chaînes :



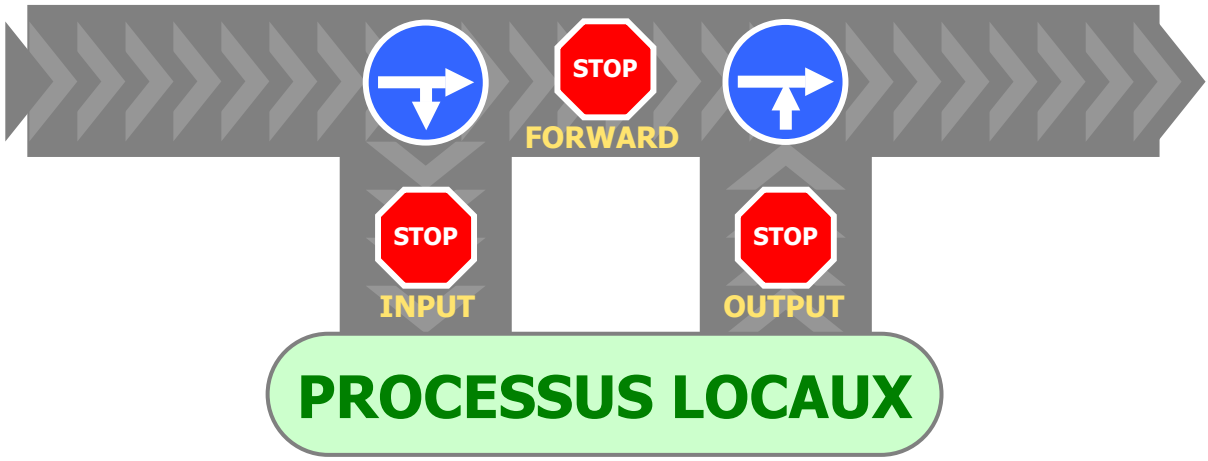
- **INPUT** Cette chaîne contrôle les paquets à destination des applications.
- **OUTPUT** Elle analyse les paquets qui sortent des applications.
- **FORWARD** Elle filtre les paquets qui passent d'une interface réseau à l'autre.(Les paquets de ce type ne passent jamais par les chaînes INPUT et OUTPUT)

La philosophie du filtrage est très simple !

Tout ce qui n'est pas explicitement autorisé est strictement interdit !

Pour cela, nous allons travailler en deux temps :

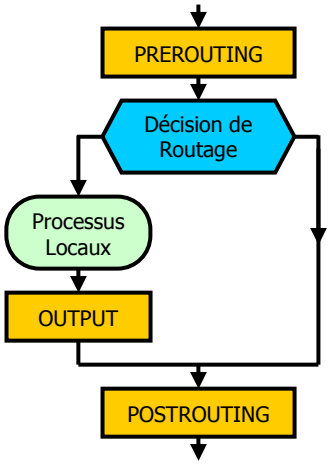
- ***Premièrement*** , **interdire par défaut tous les paquets**. C'est facile à faire, car les 3 chaînes que nous utilisons (INPUT, OUTPUT et FORWARD) ont une valeur par défaut. Donc **par défaut**, nous allons supprimer toutes les trames (on utilisera par la suite le terme de "**DROP**").
- ***Dans un second temps***, nous n'allons **autoriser que certains flux bien particuliers**. Ce sera un juste équilibre entre la sécurité du système et les fonctionnalités dont nous avons besoin.



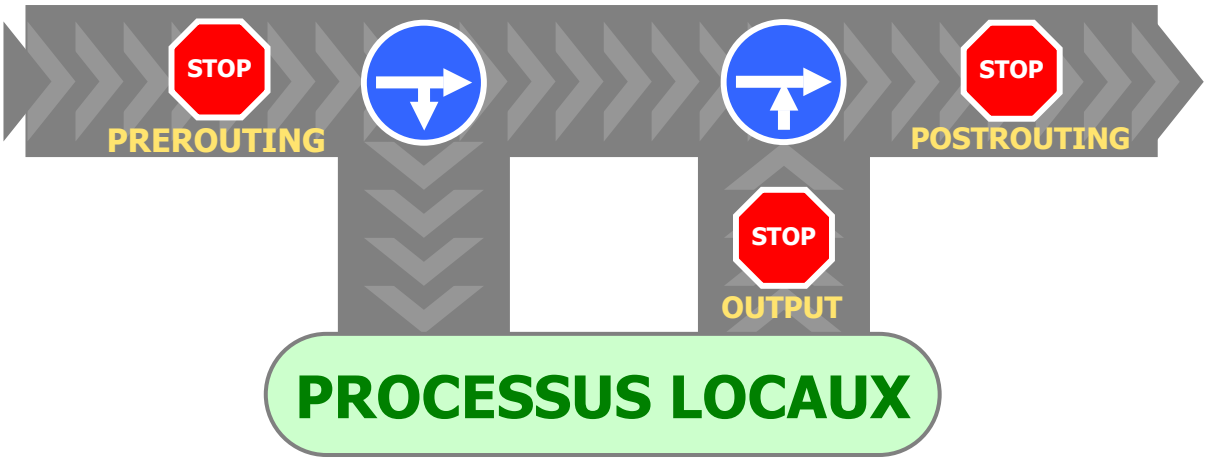
B. La table NAT

La table NAT (Network Adress Translation, ou Traduction d'Adresses Réseau) nous fait sortir du cadre orienté strictement sur la sécurité de ce document. Mais ce qu'elle permet est tout bonnement essentiel pour transformer notre machine Linux (*Phoenix*) en une **passerelle Internet**, ce qui permettra à *Paradise.* de surfer sur Internet. Et dans un second temps, nous verrons comment faire suivre certains paquets arrivant sur *Phoenix*, pour les transmettre à *Paradise*.

Pour faire tout ceci, nous avons besoin là encore de 3 chaînes :



- **PREROUTING** Les paquets vont être modifiés à l'entrée de la pile réseaux, et ce, qu'ils soient à destination des processus locaux où d'une autre interface.
- **OUTPUT** Les paquets sortant des processus locaux sont modifiés.
- **POSTROUTING** les paquets qui sont près à être envoyés aux interfaces réseaux sont modifiés.



C. La table MANGLE

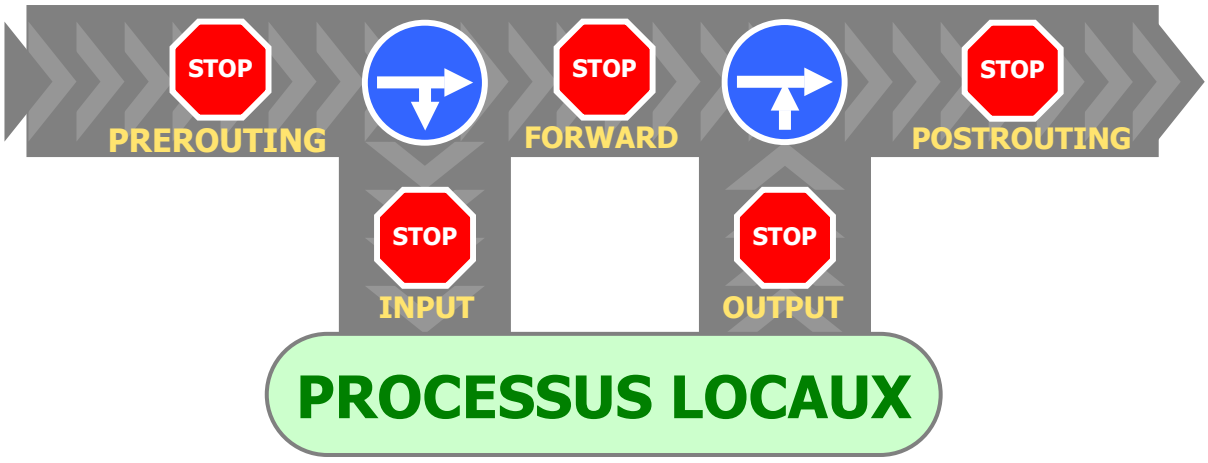
Le terme anglais "mangle" veut dire "mutilation" : Il s'agit en fait de **marquer les paquets** en entrée de la couche réseau, afin que d'autres programmes de l'espace noyau puissent en faire quelque chose. L'idée de cette technique est par exemple de fournir à Linux la possibilité d'avoir un contrôle sur les débits des flux de données entrants et sortants de la machine, afin de rendre certains flux plus prioritaires que d'autres. Certains flux vont être volontairement privilégiés, afin de garantir un meilleur confort d'utilisation à l'utilisateur. Cette technique s'appelle le QoS ("Quality of Service" ou "Qualité de Service" en français).

Un exemple d'intérêt du QoS est le suivant : Pendant que vous téléchargez en FTP un gros fichier, vous pouvez désirer continuer à surfer tranquillement sur Internet, sans être tout le temps pénalisé par votre gros téléchargement. Ceci sera possible avec le QoS, et rendant le contenu HTTP (port source 80) prioritaire par rapport à celui de votre téléchargement (port source 20).

Mais l'application de cette technique sort complètement du cadre de cette documentation.

Dans les premiers noyaux de la série 2.4, la table Mangle n'utilisait que 2 chaînes (PREROUTING et OUTPUT). Mais depuis le noyau 2.4.18 elle utilise toutes les chaînes de Netfilter :

- **PREROUTING** Les paquets vont être marqués en entrée de la couche réseau, en fonction de certains critères, de type de service (grâce aux numéros de ports source et/ou de destination), d'adresses IP de source et/ou de destination, de taille des paquets, etc. Ces informations seront utilisés par un programme fonctionnant dans l'espace noyau.
- **INPUT** Les paquets sont marqués juste avant d'être envoyés aux processus locaux.
- **FORWARD** Les paquets passant d'une interface réseau à l'autre sont marqués.
- **OUTPUT** Là, ce sont les paquets générés par les applications locales (un client web par exemple) qui vont être marqués, tout comme les paquets entrant dans la couche réseau.
- **POSTROUTING** Les paquets prêt à être envoyés sur le réseau sont marqués. L'utilisation de cette chaîne dans la table Mangle n'est cependant pas très évident.



III. Chaînes, règles et iptables

Maintenant passons à la pratique. Voyons comment nous pouvons remplir les chaînes, afin de paramétrer ces tables, et enfin sécuriser notre réseau.

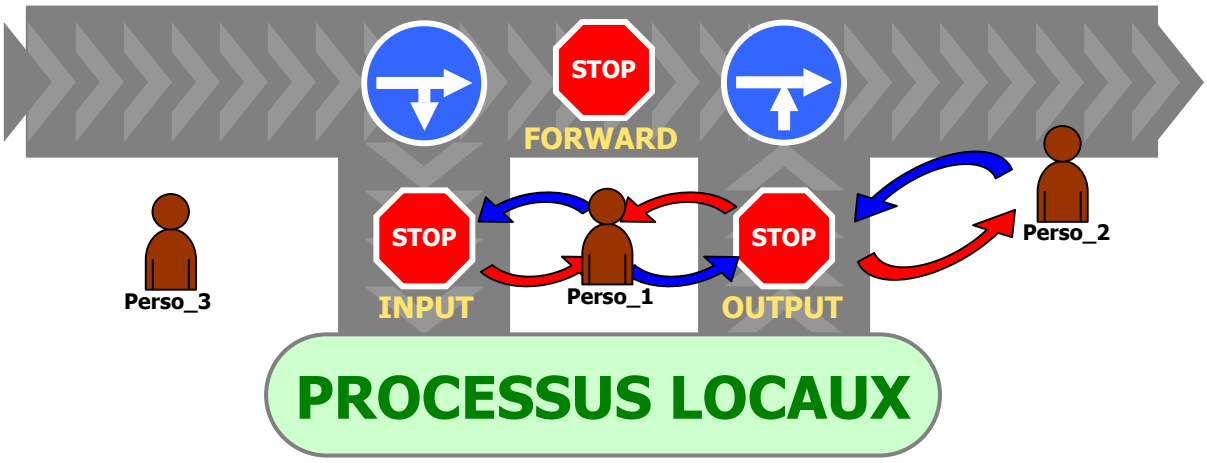
A. Les chaînes utilisateurs

Nous avons vu précédemment qu'il existe 5 chaînes principales, appelées aussi **chaînes pré-définies** , (**PREROUTING, INPUT, FORWARD, OUTPUT** et **POSTROUTING**). Ce sont les signes "STOP" des illustrations. Et nous avons aussi vu que ces chaînes sont constituées d'un certain nombre de règles, qui forment une "check list" que Netfilter parcourra afin de décider ce qu'il doit faire des paquets.

Il est possible à l'utilisateur root de **créer ses propres chaînes.** Ce sont les **chaînes utilisateurs**. Il s'agit simplement de rajouter votre propre "check list" lors de l'un des contrôles.

Sur le dessin ci-dessous, on voit quelques chaînes utilisateurs qui sont crées pour la table "Filter". Comme vous le voyez, les chaînes utilisateurs peuvent :

- Êtres utilisées par une chaîne en particulier : perso_2
- Êtres appelées par plusieurs chaînes : perso_1
- Ne pas êtres appelées du tout (perso_3). Elle ne sert à rien pour l'instant ...



B. Règles et cibles

Les règles, comme leur nom l'indique, sont une série de critères auquel doivent ou non répondre les paquets. En fait, on peut associer une règle à un "délit de sale gueule". Si le paquet réseau ressemble à l'un ou l'autre des critères, alors la règle est appliquée. Les différentes règles d'une chaînes sont appliquées les unes à la suite des autres.

Les critères peuvent être multiples :

- **Interface source ou destination.**
- **Adresse IP source ou de destination.**
- **Port source ou de destination.**
- **Type de trame.**
- **Nombre de paquets.**
- **Paquet marqué par la table Mangle.**

  Etc.

Il peut y avoir **autant de règles que l'on veut dans une chaîne**, mais il est intéressant de limiter au maximum leur nombre, afin d'avoir une vue claire et précise de notre système de filtrage.

Enfin, à **chaque règle est associée une action** (ou "CIBLE" dans la nomenclature de Netfilter) à effectuer si la règle doit s'appliquer. C'est là que Netfilter agit, qu'il fait (enfin) quelque chose avec le paquet réseau.

Les principales actions sont :

- **DROP** **Le paquet est détruit** purement et simplement. C'est typique du "délit de sale gueule"...
- **ACCEPT** Le paquet a une "bonne tête", **il est donc autorisé à continuer de passer**. Mais une autre règle située après la règle qui a accepté ce paquet peut très bien finalement décider de le supprimer.
- **REJECT** Ceci est utilisé pour **renvoyer un paquet erroné en réponse** au paquet qui correspond : sinon cela est équivalent à **DROP**. Cette cible est uniquement valable avec les chaînes **INPUT**, **FORWARD** et **OUTPUT.**
- **LOG** **Le paquet est autorisé à continuer de passer, mais ses caractéristiques sont notées au passage**. En général, c'est qu'on estime que le paquet est "louche", et que l'on veut en prendre note. Mais plus souvent encore, on décidera de le supprimer. Car, en informatique, tout ce qui est louche est anormal, et tout ce qui est anormal doit être supprimé !
- **ULOG** Cette cible fournit un espace utilisateur pour la journalisation des paquets qui on vérifié des cibles. Lorsque cette cible est positionnée pour une règle, le noyau Linux va faire un multicast du paquet par l'intermédiaire d'un socket *netlink*
- **SNAT** Cette cible n'est valable que dans la table **nat** , dans la chaîne **POSTROUTING,** Elle spécifie que l'adresse de source du paquet doit être modifiée.
- **DNAT** Cette cible n'est valable que dans la table **nat** , dans les chaînes **PREROUTING** et **OUTPUT** , et dans les chaînes définies par l'utilisateur qui sont appelées par celles ci.
- **REDIRECT** Cette cible n'est valide qu'avec la table **nat** , dans les chaînes **PREROUTING** et **OUTPUT** , et les chaînes définies par l'utilisateur qui sont appelées par celles ci. Elle modifie l'adresse IP de destination pour envoyer le paquet vers la machine elle même.
- **MASQUERADE** **Le paquet va être modifié**, afin de dissimuler (de masquer en fait) certaines informations concernant son origine. Cette technique sera utilisée un peu plus loin.
- **MARK** **Le paquet est marqué en y attachant une information**. Ceci est principalement utilisé avec les tables "Mangle", donc nous n'y reviendrons pas dessus.
- **TOS** Ceci est utilisé pour positionner le champ sur 8 bits du Type de Service dans l'en-tête IP. C'est valable uniquement lorsque vous utilisez la table **mangle.**
- **TTL** Cette cible est utilisée pour modifier le champ TTL de l'en-tête IP. Elle n'est valide que dans la table **mangle.**
- **QUEUE** permet de passer le paquet à une application, si jamais aucune application n'attend après le paquet, il est rejeté
- **RETURN** se comporte différemment selon son emplacement. S'il est dans une chaîne que vous avez construite, le paquet est renvoyé à la chaîne forgée d'où il vient. S'il se trouve déjà dans une chaîne forgée, c'est la police de cette même chaîne qui décide du sort du paquet.

Dans le cas d'une chaîne utilisateur, le paquet est envoyé à une chaîne définie par l'utilisateur, où il passera à travers de nouvelles règles. Ceci est illustré par les flèches rouges dans ce schéma ci. Si aucune décision n'est prise à la fin de la chaîne utilisateur, le paquet revient dans la chaîne courante, et passe à la règle suivante. C'est ce qu'indiquent les flèches bleues de ce même schéma .

C. Iptables

Iptables est donc une commande que seul le root peut lancer. Son but est de dialoguer avec Netfilter, afin de contrôler les règles des chaînes, dans le but de configurer les tables.

Iptables est la boîte de dialogue de Netfilter. Cette commande va pouvoir :

- Rajouter des règles / chaînes.
- Supprimer des règles / chaînes.
- Modifier des règles / chaînes.
- Afficher les règles / chaînes

Comme toute commande Linux qui se respecte, "iptables" est un programme qui se lance en ligne de commande, et qui attend de nombreux paramètres. Les lister tous n'est pas forcément très intéressant (et puis la commande " man iptables" doit bien servir à quelque chose, non ?), aussi ne verrons nous que les options les plus intéressantes.

Pour obtenir de l'aide sur une extension, utilisez l'option pour la charger ('-p', '-j' ou '-m' …) suivi de '-h' ou '--help'.

Par exemple:

```
# iptables -p tcp --help
```

Option	Paramètre	Paramètre optionnel	Explication	Exemple
-t (table)	"filter", "nat", "mangle"	Oui	Indique sur quelle table nous voulons travailler. Si aucun paramètre n'est fourni, c'est la table filter qui est sélectionnée par défaut. **Par la suite, si aucun paramètre "-t" n'est utilisé dans une commande "iptables", c'est que cette commande est destinée à la table filter.**	"**iptables -t nat …**" permet de travailler sur ta table "NAT".
-A (Append)	[Chaîne]	Non	Ajoute une règle à une chaîne prédéfinie ou utilisateur. Nous allons utiliser majoritairement cette commande.	"**iptables -A INPUT …**" ajoute une règle à la table des paquets entrant dans l'espace des programmes.
-D (Delete)	[Chaîne] [Numéro de règle]	Non	Supprime une règle dans une chaîne particulière. Le numéro de la règle peut être retrouvé avec la commande "iptables -L" ou "iptables -L -n"	"**iptables -D OUTPUT 1 -t mangle**" supprime la 1ère règle de la chaîne "OUTPUT" de la table "Mangle".
-R (Replace)	[Chaîne] [Numéro de règle]		Remplace une règle dans la chaîne sélectionnée. Si la source et/ou la destination résolvent des adresses multiple, la commande échouera. Les règles sont numérotées en partant de 1.	"**iptables -R OUTPUT 1 -t mangle**" Remplace la 1ère règle de la chaîne "OUTPUT" de la table "Mangle".
-I (Insert)	[Numéro de règle]		Insère une ou plusieurs règles dans la chaîne sélectionnée à la position donnée par le numéro de règle. Si le numéro de règle est 1 la ou les règle(s) sont insérées au début de la chaîne, c'est le comportement par défaut si aucun numéro de règle n'est spécifié	
-L (Liste)	[Chaîne]	Oui	Affiche la liste des règles pour la chaîne indiquée. Ou, si aucune chaîne n'est indiquée, cela affichera les règles pour toutes les chaînes de la table indiquée. Note : Rajouter à "-L" les options "-n" et "-v" est très utile.	"**iptables -L -n -v**" affiche le maximum d'informations sur les règles de la table "filter".
-F (Flush)	"filter", "nat", "mangle"	Oui	Supprime toutes les règles d'une chaîne prédéfinie (tel "PREROUTING", "INPUT", "FORWARD", "OUTPUT" et "POSTROUTING"). Si aucun paramètre n'est donné, toutes les chaînes prédéfinies sont supprimés.	"**iptables -F**" supprime toutes les chaînes prédéfinies de la table "FILTER".
-Z (zero)			Met à zéro le compteur de paquets et d'octets dans toute les chaînes. Il est légal de spécifier l'option **-L**, pour voir la valeur des compteurs juste avant qu'ils ne soient remis à zéro	
-N (New)	[Chaîne utilisateur]	Non	Cette option crée une nouvelle chaîne utilisateur. Par la suite, des règles doivent êtres créées, afin de remplir cette chaîne. Sinon, elle ne sera pas très utile !	"**iptables -N LogAndDrop**" crée une chaîne utilisateur dont le nom laisse supposer que l'on va logger les paquets, puis les supprimer.
-X (eXclude)	[Chaîne utilisateur]	Oui	Supprime une chaîne utilisateur. Si il n'y a aucun paramètre, toutes les chaînes utilisateurs sont supprimées.	"**iptables -X perso_3**" supprime la chaîne utilisateur "perso_3".
-P (Policy)	"filter", "nat", "mangle"	Non	Définie la politique (ou cible) par défaut d'une chaîne. Seules les chaînes prédéfinies peuvent avoir un comportement par défaut. Cette cible ne sera appliquée qu'après l'exécution de la dernière règle de la chaîne.	"**iptables -P INPUT DROP**" supprime par défaut tout les trames dans la chaîne "INPUT". Si aucune règle plus "souple" n'est définie, aucun paquet réseau n'arrivera aux processus utilisateurs de notre machine. C'est efficace comme technique, mais peu fonctionnel.
-E (Rename)	[Chaîne utilisateur]		Renomme la chaîne utilisateur spécifiée par le nom fourni. Ceci ne modifie en rien la structure de la table.	

-j (jump)	[Cible]	Non	Défini l'action à prendre si un paquet répond aux critères de cette règle. Les principales valeurs sont : ACCEPT, DROP, REJECT, LOG	"**iptables -A OUPUT -j DROP**" supprime tous les paquets sortant des processus locaux.

-i (input)	[Interface réseau]	Non	Critère sur l'interface réseau dont provient le paquet	"**iptables -A INPUT -i eth0 ...**" filtre les paquets arrivant aux programmes et venant de l'interface réseau eth0.
-o (output)	[Interface réseau]	Non	Critère sur l'interface réseau d'où les paquets vont sortir	"**iptables -A OUTPUT -o ppp0**" filtre les paquets créés par les programmes et sortant sur Internet.
-s (source)	[!] [Adresse IP ou réseau]	Non	Critère sur l'adresse IP source du paquet	"**iptables -t nat -A PREROUTING -i 192.168.0.0/24 ...**" travaille sur le routage des paquets du réseau interne.
-d (destination)	[!] [Adresse IP ou réseau]	Non	Critère sur l'adresse IP de destination du paquet	"**iptables -A OUTPUT -d 192.168.0.0/24 ...**" filtre les paquets sortant de l'espace utilisateur, à destination du réseau local.
-p (protocole)	[!] "all", "tcp", "udp", "icmp", ou un numéro	Non	Critère sur le type de trames utilisé dans le paquet. D'autres numéros de protocoles peuvent être utilisés. Voir votre fichier "/etc/procoles"	"**iptables -A INPUT -p udp ...**" filtre uniquement les paquets de type UDP.
[!] -f		Non	Cela signifie que la règle s'applique à partir du deuxième fragment d'un paquet fragmenté. Lorsque l'argument "!" précède l'option "-f", la règle ne s'appliquera qu'aux fragments de l'en-tête, ou aux paquets non fragmentés.	
--sport	[!] [Port ou service]	Non	Critère sur le port source des paquets IP	"**iptables -A INPUT -sport 80 ...** " filtre tout ce qui vient du port HTTP (80).
--dport	[!] [Port ou service]	Non	Critère sur le port de destination des paquets IP	"**iptables -O OUPUT -dport ! 21 ...** " filtre tout ce qui n'est pas à destination du FTP (21).
-m (module)	[Nom d'un module]	Non	Demande d'utiliser un module particulier	"**iptables ... -m state ...**" utilise le module de suivi de connexion ("state" veut dire "statut" en français).
--state	[!] "NEW", "ESTABLISHED", "RELATED", "INVALID"	Non	Cette option ne s'utilise que cumulée avec l'option "-m state", afin d'être utilisé pour le suivi de connexion	"**iptables ... --state NEW,ESTABLISHED ...**" filtre les paquets de nouvelles connexions, ou de connexions déjà établies via une " poignée de main ".
--limit	*taux*		Taux maximum de correspondance : spécifié par un nombre, avec un suffixe optionnel `/second', `/minute', `/hour', ou `/day' ; la valeur par défaut est 3/hour	
--log-prefix --ulog-prefix	[Un mot]	Non	Rajoute un commentaire pour les cibles LOG et ULOG	"**iptables ... -j LOG --log-prefix toto**" rajoutera le mot "toto" au log de cette règle.

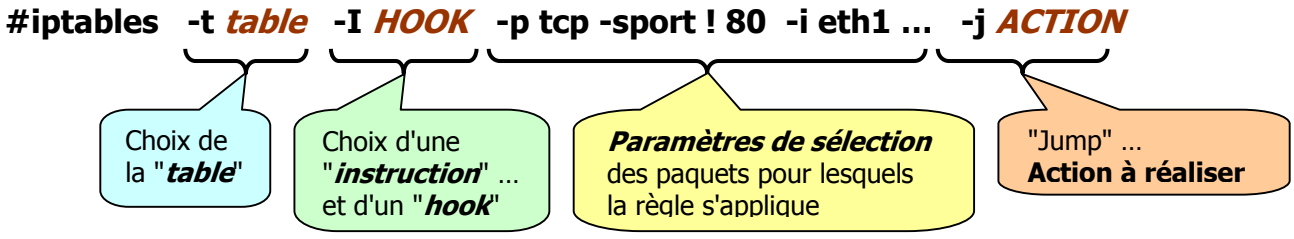
Remarque : On trouve parfois dans la colonne "Paramètre" du tableau le paramètre " [!]". Ce paramètre peut se rajouter aux autres paramètres de l'option, afin d'indiquer la négation . Exemples :

- "iptables -I INPUT -p tcp -sport 80 -j DROP"
  Supprime toutes les trames HTTP rentrant dans l'espace utilisateur.

- "iptables -I INPUT -p tcp -sport ! 80 -j DROP"
  Supprime toutes les trames rentrant dans l'espace utilisateur, excepté celles de HTTP.

Dans la suite de ce document, je vous conseille de revenir régulièrement sur ce tableau, afin de bien comprendre le mécanisme des règles. Lorsque ce tableau aura répondu à toutes vos questions, et qu'il ne suffira plus à répondre à votre soif de connaissances, je vous invite à taper un petit "man iptables" !!

La structure générale d'une ligne de commande iptable est la suivante:



D. Script Iptable basique

Le paramétrage de notre firewall va se résumer à taper un certain nombre de commandes "iptables". Et comme les règles de Netfilter sont perdues à chaque arrêt de la machine, il faudrait tout recommencer à chaque fois que vous démarrerez votre ordinateur. Il y a un moyen simple de d'éviter cela avec des scripts.

Nous allons commencer par travailler exclusivement sur la table filter.

Bon, petit rappel sur LA règle de filtrage universellement reconnue :

- **Premièrement** , vider toutes les chaînes de toutes les tables de Netfilter, afin de savoir exactement ce que l'on a dans notre firewall. Mais il ne faudra pas rester trop longtemps dans cette situation, car la machine sera sans aucune protection.

```
iptables -F
iptables -X
```

- **Deuxièmement** , interdire par défaut tous les paquets. Pour cela, nous allons utiliser l'option "-P" ("Politique par défaut) des chaînes INPUT, FORWARD et OUTPUT de la table filter.

- **Dans un dernier temps**, nous n'allons autoriser que certains flux bien particuliers.

- Concernant **l'ordre des règles** : Lorsque nous appelons la commande "iptables" pour rajouter/supprimer des règles, l'ordre d'insertion de celles-ci à une certaine importance. Si une première règle supprime un certain type de paquets, une seconde règle écrite un peu plus loin dans votre script ne verra jamais ces paquets. Donc inutile de les accepter, ou de les supprimer de nouveau. Mais en général, comme on écrit uniquement des règles pour accepter des paquets ("-j ACCEPT"), il n'y pas d'importance dans l'ordre des règles.

Bon, gardez en mémoire le tableau des options d'iptables et mettons nous au boulot :
Suppression de toutes les chaînes : C'est facile, il faut supprimer les tables pré-définies (option " -F") et toutes les tables utilisateurs (option "-X").

On supprime tout :

```
[root@phoenix /]# iptables -t filter -F
[root@phoenix /]# iptables -t filter -X
```

Définition de la politique (cibles) par défaut : La table **filter** possède 3 chaînes, donc elles sont toutes les trois à initialiser. Par défaut, on décide donc de tout détruire (**DROP**) :

```
[root@phoenix /]# iptables -t filter -P INPUT DROP
[root@phoenix /]# iptables -t filter -P OUTPUT DROP
[root@phoenix /]# iptables -t filter -P FORWARD DROP
```

A ce stade là, vous avez court-circuité tout votre système de réseau. Toutes vos connexions réseaux sont hors service. Pas un logiciel que vous utilisez ne peut accéder au réseau, ou à vous propres serveur. J'en veux pour preuve, la commande "ping localhost" ne marche même plus, et pourtant il lui en faut pour ne plus marcher !

```
[coco@phoenix /]$ ping localhost
PING localhost.local.net (127.0.0.1) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted

--- localhost.local.net ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2011ms
```

C'est exactement ce que nous voulions faire. D'un autre coté, c'est la protection absolue du réseau, qui rivalise presque avec la déconnexion physique des câbles... Mais quand à l'intérêt de la chose, c'est plutôt limité !

Autoriser quelques connexions : Dans notre réseau, nous avons **2 cartes réseaux** ("**eth0**" et "**eth1**"), ainsi que **l'interface de loopback** ("**lo**"). Occupons nous donc de chacune des 3 interfaces :

1. >> lo (réseau virtuel localhost) :

C'est le plus facile !

Nous pouvons avoir toute confiance en ce réseau, car il est interne à la mémoire de notre machine. Nous allons donc autoriser ("-j ACCEPT") toutes les connexions sortantes des processus locaux ("-A OUTPUT") par cette interface virtuelle ("-o lo") ayant une adresse de loopback ("-s 127.0.0.0/8"), et à destination des machines de ce réseau virtuel (-d "127.0.0.0/8) :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o lo -s 127.0.0.0/8 -d 127.0.0.0/8 \
-j ACCEPT
```

Puis nous allons faire l'inverse, c'est à dire autoriser ("**-j ACCEPT**") toutes les connexions rentrantes dans les processus locaux ("**-A INPUT**"), par cette interface virtuelle ("**-i lo**"), venant des machines de ce réseau virtuel ("**-s 127.0.0.0/8**") et à destination des adresses de loopback ("**-d 127.0.0.0/8**") :

```
[root@phoenix /]# iptables -t filter -A INPUT -i lo -s 127.0.0.0/8 -d 127.0.0.0/8 -j ACCEPT
```

Et nous pouvons immédiatement vérifier que le "ping localhost" fonctionne à nouveau :

```
[coco@phoenix /]$ ping localhost
PING localhost.local.net (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.local.net (127.0.0.1): icmp_seq=1 ttl=64 time=0.134 ms
64 bytes from localhost.local.net (127.0.0.1): icmp_seq=2 ttl=64 time=0.091 ms

--- localhost.local.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.091/0.112/0.134/0.023 ms
```

2. >> eth0 (réseau interne "local.net") :

Cela reste encore facile.

Nous désirons dans un premier temps permettre à paradise.local.net de dialoguer sans limite avec phoenix0.local.net. Quoi ? Sans limite ? Mais ce n'est pas un peu dangereux, non ? Certes cela peu se discuter, mais comme à priori nous travaillons dans un réseau local et qui plus est dans le réseau local d'un particulier, nous pouvons avoir une certaine confiance dans les machines de notre propre réseau, non ? Ah, vous utilisez Windows® sur les autres machines de votre réseau ? Et vous avez peur que ce Windows agresse vos Linux ? En allant un peu plus loin dans ce document, vous comprendrez comment on peut mettre un filtrage un peu plus restrictif, donc n'anticipons pas.

Donc nous avons fait comme pour l'interface "lo", et autoriser les processus locaux à dialoguer en entrée et en sortie avec le réseau local :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o eth0 -s 192.168.0.0/24 -d 192.168.0.0/24
                  -j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.0/24 -d 192.168.0.0/24 -
                  j ACCEPT
```

Bien, et que donne le "ping paradise.local.net" ?

```
[coco@phoenix /]$ ping paradise.local.net
PING paradise.local.net (192.168.0.2) 56(84) bytes of data.
64 bytes from paradise.local.net (192.168.0.2): icmp_seq=1 ttl=64 time=0.134 ms
64 bytes from paradise.local.net (192.168.0.2): icmp_seq=2 ttl=64 time=0.091 ms

--- localhost.local.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.091/0.112/0.134/0.023 ms
```

3. >> eth1 (réseau Internet "internet.net") :

là, c'est la plus compliquée...

Que vous nous faire exactement en fait ? **Interdire l'accès à pirate.internet.net**, mais autoriser celui de **web.internet.net**. C'est facile alors, il suffit de mettre uniquement des règles "-j ACCEPT" en direction et depuis l'adresse IP de web.internet.net ? En théorie oui... Mais est-ce que vous connaissez l'adresse IP de tous les intrus qui veulent vous ennuyer ? Non hein ? Ce n'est pas le genre d'informations qui se trouve sous le sabot d'un cheval ! Et puis en plus, quelqu'un sur web.internet.net pourrait héberger sans que vous le sachiez un autre intrus... Que la vie est compliquée, non ?

En fait, c'est très simple : nous n'allons autoriser que les connexions vers les services web qui nous intéresse, à savoir le **HTTP (port 80 en TCP)**, et le **HTTPS (443 en TCP)**, et cela pour toutes les machines. Seul les requêtes en direction et venant des ports seront autorisées. Hop, 4 règles d'"iptables" est c'est fait :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o eth1 -s 10.0.0.0/8 -p tcp --dport 80 \
-j ACCEPT
[root@phoenix /]# iptables -t filter -A OUTPUT -o eth1 -s 10.0.0.0/8 -p tcp --dport 443 \
-j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -d 10.0.0.0/8 -p tcp --sport 80 \
-j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -d 10.0.0.0/8 -p tcp --sport 443 \
-j ACCEPT
```

Il ne nous reste plus qu'à faire ouvrir une page web sur **web.internet.net**, ou de faire un petit "**nmap**" sur ses ports 80 et 443, afin de voir si tout marche bien.

```
[coco@phoenix /]$ nmap web.internet.net -p 80,443

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on web.0.0.10.in-addr.arpa (10.0.0.200):
Port State Service
80/tcp open http
443/tcp open https

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Mais en fait il y a un truc important qui ne marche pas... Depuis Phoenix, essayez de faire un "ping phoenix.local.net", un "ping phoenix0.local.net", ou un "ping phoenix1.local.net" :

```
[coco@phoenix /]$ ping phoenix.local.net
PING phoenix.local.net (192.168.0.1) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted

--- phoenix.local.net ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1010ms

[coco@phoenix /]$ ping phoenix0.local.net
PING phoenix0.local.net (192.168.0.1) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted

--- phoenix0.local.net ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1019ms

[coco@phoenix /]$ ping phoenix1.internet.net
PING phoenix1.internet.net (10.0.0.1) 56(84) bytes of data.
ping: sendmsg: Operation not permitted

--- phoenix1.internet.net ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Pourquoi cela ne marche t'il pas ?

Souvenez vous lors du 1er chapitre, nous en avions rapidement parlé.

En fait, c'est très simple : dans nos règles "iptables" de configuration de localhost, nous avons indiqué que nous autorisions uniquement les échanges sur l'interface localhost, pour les machines du réseau 127.0.0.0/8. Hors, et c'est ce que nous avions vu lors du 1er chapitre, lorsque qu'une machine accède à ses propres ressources elle ne passent pas par l'interface physique, mais uniquement par l'interface loopback. Donc au lieu de passer par les règles de l'interface eth0, ces paquets sont passés par ceux de l'interface lo, et se sont fait détruire par la règle DROP par défaut... C'est très fort, non ?

Comment résoudre ce problème alors ? La solution est très simple, il suffit d'être un peu plus "tolérant" sur les 2 règles de l'interface "localhost", et retirer les options "-d 127.0.0.0/8" et "-s 127.0.0.0/8". Pour cela, nous allons supprimer les précédentes règles, et les recréer :

```
[root@phoenix /]# iptables -t filter -L -n -v
Chain INPUT (policy DROP 463 packets, 35458 bytes)
   pkts  bytes  target   prot  opt   in   out      source      destination
   0        0    ACCEPT    all   --   lo    *    127.0.0.0/8    127.0.0.0/8    <-Règle 1
   0        0    ACCEPT    all   --  eth0   *   192.168.0.0/24   0.0.0.0/0
   0        0    ACCEPT    tcp   --  eth1   *     0.0.0.0/0      0.0.0.0/0     tcp spt:80
   0        0    ACCEPT    tcp   --  eth1   *     0.0.0.0/0      0.0.0.0/0     tcp spt:443

Chain FORWARD (policy DROP 0 packets, 0 bytes)
   Pkts  bytes  target   prot  opt   in   out      source      destination

Chain OUTPUT (policy DROP 71 packets, 10712 bytes)
   Pkts  bytes  target   prot  opt   in   out      source      destination
   0        0    ACCEPT    all   --    *    lo    127.0.0.0/8    127.0.0.0/8    <-Règle 1
   0        0    ACCEPT    all   --    *   eth0    0.0.0.0/0   192.168.0.0/24
   0        0    ACCEPT    tcp   --    *   eth1    0.0.0.0/0      0.0.0.0/0     tcp dpt:80
   0        0    ACCEPT    tcp   --    *   eth1    0.0.0.0/0      0.0.0.0/0     tcp dpt:443

[root@phoenix /]# iptables -t filter -D OUTPUT 1
[root@phoenix /]# iptables -t filter -D INPUT 1
[root@phoenix /]# iptables -t filter -A OUTPUT -o lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT
[root@phoenix /]# iptables -t filter -L -n -v
Chain INPUT (policy DROP 475 packets, 37048 bytes)
   Pkts  bytes  target   prot  opt   in   out      source      destination
   1      248    ACCEPT    all   --  eth0   *   192.168.0.0/24   0.0.0.0/0
   0        0    ACCEPT    tcp   --  eth1   *     0.0.0.0/0      0.0.0.0/0     tcp spt:80
   0        0    ACCEPT    tcp   --  eth1   *     0.0.0.0/0      0.0.0.0/0     tcp spt:443
   274    166K   ACCEPT    all   --   lo    *     0.0.0.0/0      0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts     bytes  target   prot  opt   in   out      source      destination

Chain OUTPUT (policy DROP 84 packets, 12080 bytes)
   pkts  bytes  target   prot  opt   in   out      source      destination
   9      872    ACCEPT    all   --    *   eth0    0.0.0.0/0   192.168.0.0/24
   0        0    ACCEPT    tcp   --    *   eth1    0.0.0.0/0      0.0.0.0/0     tcp dpt:80
   0        0    ACCEPT    tcp   --    *   eth1    0.0.0.0/0      0.0.0.0/0     tcp dpt:443
   274    166K   ACCEPT    all   --    *    lo     0.0.0.0/0      0.0.0.0/0
```

Cette fois ci, un "**ping phoenix0.local.net**" et des autres adresses IP de Phoenix marche sans problème.
Vous vous demandez sûrement pourquoi j'ai introduit cette "erreur" dans mon premier script ? C'est simple :

- Il sera très important par la suite de comprendre cette histoire d'auto connexion.
- Je ne suis pas sûr que vous vous en seriez souvenu si je ne vous avais pas fait pointer le doigt dessus.
- Au passage, nous avons vu une utilisation de deux commandes que nous ne connaissions pas : "iptables -t filter -L -n -v" pour afficher la liste des chaînes d'une table, et "iptables -t filter -D ...." qui permet de supprimer une règle. Comme quoi, les erreurs sont parfois constructives !

Bon, pendant que vous prenez un biscuit apéro bien mérité, je ferais quelques remarques sur ces scripts :

- Dans toutes nos commandes, nous avons utilisé l'option **"-t filter"**, pour indiquer que nous voulions travailler sur cette table. C'est très bien, mais c'est un peu inutile. En effet, par défaut "**iptables**"

considère que c'est la table "**filter**" qui est utilisée. Donc par la suite, on pourra économiser de la ligne de commande en n'utilisant plus le "-**t filter**" pour ces commandes **iptables** là.

- Dans nos commandes **iptables** ayant pour cible "**ACCEPT**", nous avons cherché à rajouter le maximum de paramètre , en combinant par exemple les options "**-i**" avec "**-s**". C'est une bonne chose, car les règles d'acceptation des paquets doivent toujours être les plus restrictives possible. Ne rentre pas sur notre ordinateur qui veut ! Même si au passage, un peu trop de rigueur amène à des erreurs du type du "**localhost**"...
- Même si nos règles sur le réseau "**internet.net**" sont satisfaisantes, car bien strictes, elle sont un peu lourdes à gérer. 4 règles "**iptables**" uniquement pour autoriser la visite de sites web, sécurisés ou non, cela fait beaucoup... D'autant que nous n'avons pas parlé de FTP, de IRC, de "chat" ("discussion" en français), et que sais je encore.
- Enfin, au point où nous en sommes, nous avons l'équivalent d'un petit firewall de type matériel, que l'on peut trouver dans le commerce, voir même équivalent au firewall intégré dans Windows® XP (à ce jour, il ne gère pas le conntrack).

Cependant, nous pouvons être satisfait de notre travail. Sans protection de la part de netfilter, l'intrus pouvait voir tout nos ports ouverts :

```
[intrus@pirate /]$ nmap phoenix1.internet.net

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on phoenix1.internet.net (10.0.0.1):
(The 1590 ports scanned but not shown below are in state: closed)
Port       State      Service
21/tcp     open       ftp
23/tcp     open       telnet
25/tcp     open       smtp
53/tcp     open       domain
80/tcp     open       http
110/tcp    open       pop-3
111/tcp    open       sunrpc
139/tcp    open       netbios-ssn
443/tcp    open       https
3306/tcp   open       mysql
6000/tcp   open       X11

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

Maintenant, avec notre second script netfilter , non seulement il ne voit plus rien, mais en plus il suppose que Phoenix est arrêté :

```
[intrus@pirate /]# nmap phoenix1.internet.net

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0

Nmap run completed -- 1 IP address (0 hosts up) scanned in 30 seconds
```

Alors que nous, nous pouvons surfer en toute tranquillité

Ahhh, que c'est beau l'insouciance de ces jeunes internautes... Mais à votre avis, pourquoi il y a encore quelques paragraphes après celui-ci ? Humm ? Je vous le donne en mille : c'est à cause du retour de l'intrus masqué ! Il revient, et il n'est pas du tout content que vous ayez essayé de l'empêcher de rentrer sur votre machine... En fait, nous avons perdu cette bataille face à lui, et nous ne le savons même pas encore...

IV. Le suivi de connexion (conntrack)

A. Comment leurrer un firewall ...

Phoenix se trouve toujours dans la configuration Netfilter définie par le script "iptables-basic-2.sh", et Pirate ne nous voit toujours pas en utilisant la commande "nmap". Oui, mais notre intrus n'est pas un petit plaisantin. Comme nous, il sait lire le "man nmap", et il connaît l'option " -g"...

Notre intrus va donc se connecter en temps que root sur sa propre machine, et lancer la commande "nmap phoenix1.internet.net -g 80" :

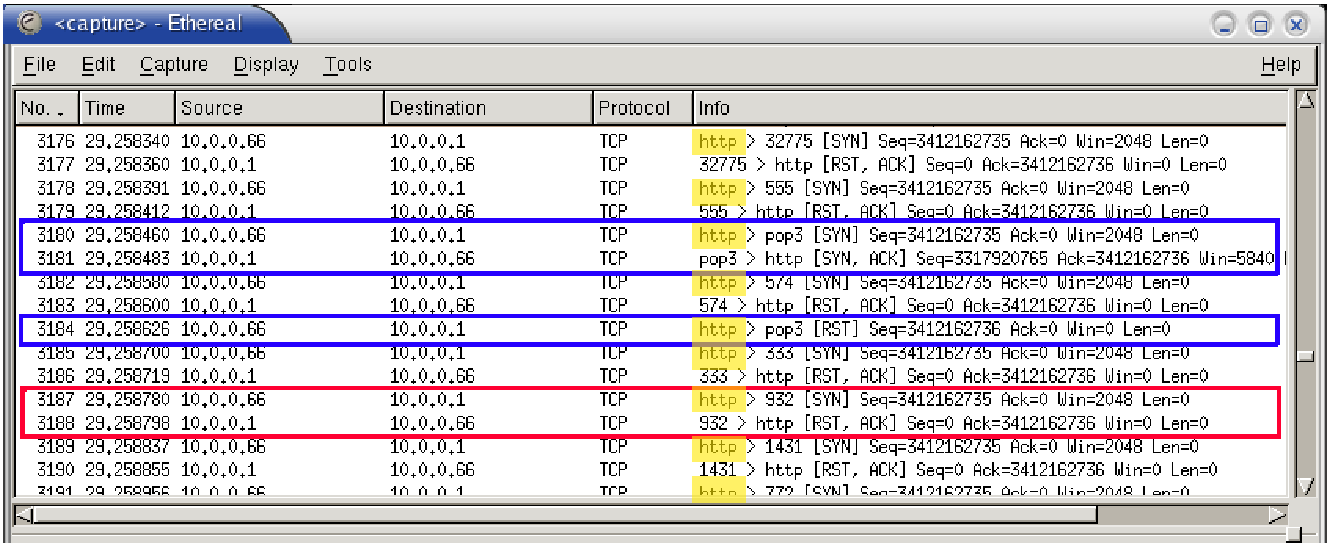
```
[root@pirate /]# nmap phoenix1.internet.net -g 80

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on phoenix1.internet.net (10.0.0.1):
(The 1585 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
53/tcp     open        domain
80/tcp     open        http
110/tcp    open        pop-3
111/tcp    open        sunrpc
139/tcp    open        netbios-ssn
307/tcp    filtered    unknown
404/tcp    filtered    nced
443/tcp    open        https
511/tcp    filtered    passgo
578/tcp    filtered    ipdd
607/tcp    filtered    nqs
3306/tcp   open        mysql
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds
```

Notre intrus a réussi à nous retrouver, mais en plus il voit tout nos ports ouverts !

En fait c'est tout à fait naturel et cela prouve la protection complètement insuffisante d'un firewall basé uniquement sur l'ouverture des ports. L'image ci-contre le montre bien. C'est une capture des connexions réseaux faite par "**ethereal**" sur **phoenix1.internet.net**. On y voit :



- En rouge : **Pirate ouvre une connexion venant du port 80**, et à destination d'un port de Phoenix sur lequel il suppose que tourne un démon. Il commence une " hand check" tout à fait classique (SYN), auquel répond Phoenix par un "RESET" (RST, ACK), disant que ce port (932) n'est pas ouvert.

- En bleu : là encore, Pirate ouvre une connexion venant du port 80. Mais cette fois ci, Phoenix l'informe que le port ("pop3" <=> 110) est ouvert. Poliment, Pirate referme la connexion (RST) afin de ne pas donner l'éveil à Phoenix. Mais au passage, il a pu noter que le port en question était ouvert... Victoire sur tout la ligne pour l'intrus...

Comme on le voit, l'utilisation du paramètre " **-g 80**" permet de configurer "**nmap**" en lui disant de se faire passer pour un serveur web ("HTTP" <=> 80). Cette option n'est utilisable que parce que l'intrus a lancé la commande en temps que root. Mais qu'importe, l'intrus est forcément root sur sa machine. Et dans notre configuration de Netfilter, comme nous avons explicitement autorisé les connexions rentrantes venant du port 80, Netfilter ne peut pas interdire ce port scanning...

Quant à interdire les connexions venant du **port 80**, c'est tout bonnement impossible, car sinon nous ne pourrions plus accéder à des sites web...

On peut se demander si un tel port scanning a un réel intérêt. Bon, l'intrus sait que nos ports sont ouverts. Oui, et alors ? La belle affaire ! Notre Netfilter est là pour nous protéger, non ? Non, malheureusement pas avec une configuration si pitoyables...

Bon, notre intrus n'est pas née de la dernière pluie, et il connaît aussi le programme "**nc**" ("nc – couteau suisse de TCP/IP", "man nc" pour plus d'informations). Avec ce programme, il va pouvoir ouvrir une connexion TCP/IP depuis son port 80, vers le port 80 de Phoenix, et envoyer toutes les commandes qu'il veut. Comme part exemple un "GET /index.html" qui permet de télécharger la page "/index.html". Cette commande est la base du protocole HTTP, et votre navigateur le fait pour ainsi dire tout le temps.

```
[root@pirate /]# nc -p 80 phoenix1.internet.net 80
GET /
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Mozilla/4.61 [en] (X11; I; Linux 2.2.9-23mdk i686)
[Netscape]">
<meta name="Author" content="MandrakeSoft">
<title>Welcome to the Advanced Extranet Server, ADVX!</title>
```

La fin de la réponse a été coupée, car inutile. On voit bien ici la réponse du serveur web de Phoenix, qui affiche tout naturellement une page web de notre réseau local... Que le lecteur continue de s'inquiéter, ce qui est fait ici très simplement avec un serveur web, peut l'être tout autant avec un serveur un peu plus sensible, comme par exemple le partage de fichier "**Samba**" ou "**NFS**". Il ne faut pas grand chose en fait, juste un peu modifier les sources de programmes comme "**smbmount**" ou "**mount**"...

Bon, si à ce stade je n'ai pas réussi à capter toute votre attention sur ce problème, vous pouvez définitivement arrêter la lecture de ce document, et continuer de vous faire allégrement pirater !

B. Et comment renvoyer l'intrus dans sa niche

Bien, le problème ici est que notre intrus peut rentrer comme il veut par les différents ports que nous aurions laissé ouverts afin que nous, nous puissions aller sur Internet. En fait, il faudrait que notre firewall soit semi-perméable, c'est à dire qu'il ne laisse entrer des connexions venant du **port 80**, que si elles viennent en réponse de connexions qui auraient été initialisées par Phoenix. C'est justement ce que permet le module de suivit de connexion (**conntrack**) de Netfilter.

Cette technique est vraiment une avancée en matière de firewall. Netfilter se base sur la poignée de main ("hand check", que nous avons vu au 1er chapitre), afin de déterminer si une connexion a été initialisée par Phoenix ou non. Pour toute nouvelle connexion sortante, il associe l'état "**NEW**" ("nouveau" en anglais) à la connexion, et stocke cette information en mémoire. Quand il verra arriver d'autres connexions venant de l'extérieur en réponse à cette connexion "**NEW**", il leur attribuera alors l'état " **ESTABLISHED**" ("établie" en anglais). Ainsi, pour une connexion rentrante et qui ne se trouve pas déjà dans la mémoire de **Netfilter**, celui-ci pourra en déduire qu'elle a été initialisée par l'extérieur, et lui attribuera l'état " **INVALID**". Dans ce cas, et si il a été configuré pour, Netfilter pourra refuser ("**DROP**") ou rejeter ("**REJECT**") cette connexion.

Il existe un dernier statut intéressant pour le suivit de connexion, il s'agit du statut " **RELATED**" : Il existe certains protocoles, comme le **FTP** ou l'**IRC** par exemple, qui ne répondent pas toujours sur le port qui a initialisé la connexion. Au lieu de cela, ils initialisent eux-mêmes une connexion sur un autre port de la machine demandant l'information. Vous pouvez trouver une excellente explication sur le fonctionnement de ce mode particulier du protocole **FTP** sur ce site (la partie sur le "**mode passif**"). Pour certains de ces protocoles particuliers, **Netfilter** est capable de comprendre que cette nouvelle connexion rentrante est en fait en relation avec une autre connexion précédemment établie par la machine elle-même. Pour ce type de connexion, elle leur attribue le statut de "**RELATED**" (pour "**relative**" en anglais).

Voyons maintenant comment mettre en oeuvre cette technique. Le système de suivit de connexion de **Netfilter** n'est pas forcément compilé dans le **kernel**, donc si ce n'est pas le cas, vous devez d'abord charger le module "**ip_conntrack**" :

```
[root@phoenix /]# modprobe ip_conntrack
```

Si vous envisagez d'utiliser vos clients **FTP en mode passif**, ou que vous compter utiliser l'**IRC**, vous avez la possibilité de charger les modules "**ip_conntrack**" qui supportent ces protocoles :

```
[root@phoenix /]# modprobe ip_conntrack_ftp
[root@phoenix /]# modprobe ip_conntrack_irc
```

Enfin, pour voir la listes des modules "**ip_conntrack**" qui sont installés sur votre machine :

```
[root@phoenix /]# uname -a
Linux phoenix.local.net 2.4.21-0.13mdksmp #1 SMP Fri Mar 14 13:41:18 EST 2003 i686 unknown
unknown GNU/Linux
[root@phoenix /]# find /lib/modules/2.4.21-0.13mdksmp/ -iname "ip_conntrack_*"
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ip_conntrack_ftp.o.gz
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ip_conntrack_proto_gre.o.gz
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ip_conntrack_h323.o.gz
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ip_conntrack_irc.o.gz
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ip_conntrack_pptp.o.gz
```

Maintenant, comment indiquer à Netfilter que nous ne désirons laisser passer que les connexions sortantes initialisées par la machine, mais aussi de ne laisser passer que celles qui sont en réponse avec les premières ? C'est tout simplement aussi facile que ce que nous venons de dire :

```
[root@phoenix /]# iptables -A OUTPUT -o eth1 -s 10.0.0.1 -d 0.0.0.0/0 -p all -m state --
state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -A INPUT -i eth1 -s 0.0.0.0/0 -d 10.0.0.1 -p all -m state --
state RELATED,ESTABLISHED -j ACCEPT
```

- **Le paramètre "-m state"** signifie que nous avons besoin du module "**state**" (celui du suivit de connexion), et de ses options particulières.
- "**--state**" indique les états qui vont être utilisés pour nos règles.
- "**! INVALID**" : comme vu plus haut, le module de suivit de connexion gère 4 états, **NEW**, **ESTABLISHED**, **RELATED** et **INVALID**. Ce paramètre indique que l'on accepte uniquement les **connexions qui ne sont pas invalides**, donc qui sont "**NEW**, **ESTABLISHED**, **RELATED**". Ceci est uniquement un manière plus rapide d'écrire qu'il faut que les connexions soient d'un des 3 états "**NEW**, **ESTABLISHED**, **RELATED**".

Voila, nous venons de terminer notre configuration de Netfilter en utilisant le suivi de connexion. Le script iptables de tout ceci est ici :

```
#!/bin/sh
###############################################################################
#
# NOM: iptable-conntrack-1.sh
# COMMENTAIRE : Exemple d'utilisation du suivi de connexion
#
###############################################################################

# Chargement des modules de suivi de connexion
modprobe ip_conntrack        ; # Module principal de suivi de connexion
modprobe ip_conntrack_ftp    ; # Module de suivi de connexion FTP
modprobe ip_conntrack_irc    ; # Module de suivi de connexion IRC

# Suppression de toutes les chaînes pré-définies de la table FILTER
iptables -t filter -F

# Suppression de toutes les chaînes utilisateur de la table FILTER
iptables -t filter -X

# Définition des règles par défaut: toute les paquets sont détruits
iptables -t filter -P INPUT DROP
iptables -t filter -P OUTPUT DROP
iptables -t filter -P FORWARD DROP

# Autorise l'interface loopback à dialoguer avec elle-même
iptables -t filter -A OUTPUT -o lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT
iptables -t filter -A INPUT  -i lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT

# Autorise les connexions avec le réseau 192.168.0.0/24 connecté à l'interface eth0
iptables -t filter -A OUTPUT -o eth0 -s 192.168.0.0/24 -d 192.168.0.0/24 -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -s 192.168.0.0/24 -d 192.168.0.0/24 -j ACCEPT

# Autorise les connexions avec internet uniquement si elles sont initialisées par
# les process locaux
iptables -t filter -A OUTPUT -o eth1 -s 10.0.0.1  -d 0.0.0.0/0 -p all -m state \
--state ! INVALID -j ACCEPT
iptables -t filter -A INPUT  -i eth1 -s 0.0.0.0/0 -d 10.0.0.1  -p all -m state \
--state RELATED,ESTABLISHED -j ACCEPT
```

Mais est-ce vraiment efficace ? On va le voir tout de suite :

```
[root@pirate /]# nmap phoenix1.internet.net -g 80 -P0

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
All 1601 scanned ports on phoenix1.internet.net (10.0.0.1) are: filtered

Nmap run completed -- 1 IP address (1 host up) scanned in 1721 seconds
```

On utilise l'option **-P0** pour forcer la connexion par "**nmap**", vu que Phoenix refuse de répondre aux demandes les plus simples. On remarque que le **scan** est excessivement long (1721 secondes, soit près d'une demi heure...), car Pirate fait son scan en testant une dizaine de ports en parallèle, mais attends une dizaine de secondes pour chaque groupe de ports testés.

Dans ces conditions, **nmap** ne peut déterminer si Phoenix est joignable ou pas.

Sur Phoenix on peut voir par contre qu'un grand nombre de paquets entrants ont été "dropés". Ce sont les tentatives de connexions de "**nmap**" :

```
[root@phoenix /]# iptables -L -v -n
Chain INPUT (policy DROP 6571 packets, 268K bytes)
pkts bytes target  prot opt  in  out      source      destination
3480 2235K ACCEPT   all  --  lo   *     0.0.0.0/0     0.0.0.0/0
9     2232  ACCEPT   all  -- eth0  *   192.168.0.0/24 192.168.0.0/24
0        0  ACCEPT   all  -- eth1  *     0.0.0.0/0      10.0.0.1    state RELATED,ESTABLISHED

Chain FORWARD (policy DROP 0 packets, 0 bytes)
```

```
pkts bytes target  prot opt  in  out      source      destination

Chain OUTPUT (policy DROP 22 packets, 1716 bytes)
pkts bytes target  prot opt  in  out      source      destination
0      0   ACCEPT  tcp  --   *   *     0.0.0.0/0     0.0.0.0/0   tcp spt:6000
0      0   ACCEPT  tcp  --   *   *     0.0.0.0/0     0.0.0.0/0   tcp spt:6000
3480 2235K ACCEPT  all  --   *   lo    0.0.0.0/0     0.0.0.0/0
9     2232 ACCEPT  all  --   *  eth0 192.168.0.0/24 192.168.0.0/24
97    9096 ACCEPT  all  --   *  eth1    10.0.0.1      0.0.0.0/0   state
NEW,RELATED,ESTABLISHED
```

La conclusion est sans appel : notre machine est un "trou noir" qui se refuse de répondre aux sollicitations extérieures . Par contre, Phoenix arrive toujours à se connecter sur internet.net :

```
[coco@phoenix /]$ ping web.internet.net
PING web.internet.net (10.0.0.200) 56(84) bytes of data.
64 bytes from web.0.0.10.in-addr.arpa (10.0.0.200): icmp_seq=1 ttl=64 time=2.22 ms
64 bytes from web.0.0.10.in-addr.arpa (10.0.0.200): icmp_seq=2 ttl=64 time=0.300 ms

--- web.internet.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.300/1.263/2.227/0.964 ms

[coco@phoenix /]$ nmap web.internet.net -p 80,443

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on web.0.0.10.in-addr.arpa (10.0.0.200):
Port       State   Service
80/tcp     open    http
443/tcp    open    https

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

L'intrus ne peut nullement accéder de lui-même à notre machine, toutes ses tentatives de "**nmap**" échoues. Votre machine a donc été en grande partie sécurisée. Comparé à la situation du début, vous pouvez donc être un peu plus tranquille, et vous avez un peu moins à craindre des intrus. Mais, et comme nous le verrons en conclusion, ce n'est pas pour autant que vous ne risquez rien.

Bien, maintenant que nous avons sécurisé l'accès à notre machine grâce à la table " **Filter**", passons aux autres fonctionnalités intéressantes de Netfilter : passons à l'exploitation de la table "**NAT**".

V. IP masquerading / Port forwarding

Si **Netfilter** est très efficace pour filtrer les connexions entrantes et sortantes des processus locaux, il peut servir aussi à d'autres fonctionnalités, comme **le masquage IP et le suivi de port**.

A. IP masquerading

Jusqu'à présent, la machine Paradise de notre réseau n'a pas été très utilisée. Son activité réseau est restreinte à celle du réseau **local.net**. Alors que Phoenix, lui, peut se connecter au réseau **internet.net**. Tout cela n'est pas très juste, aussi allons nous y remédier. Comment ? En transformant tout simplement Phoenix en une **passerelle** entre les réseaux **local.net** et **internet.net**.

Dans tout ce qui suit, nous allons nous arranger pour que **paradise.local.net** puisse se connecter au serveur web **web.internet.net**.

Sous Windows®, ce que nous allons faire s'appelle du "**partage de connexion Internet**", ce qui est tout à fait vrai : Phoenix va partager sa connexion à Internet avec les machines du réseaux local.net.
Pour réaliser ceci, il nous faut réaliser plusieurs opérations :

- ***La première***, c'est de charger les modules dont nous allons avoir besoin. En premier, nous avons besoin du module de **NAT**, c'est à dire "**iptables_nat**". Comme nous voulons aussi faire du suivi de connexion sur les paquets **NAT**, nous chargerons de même les modules **NAT FTP** et **IRC**, "**ip_nat_ftp**" et "**ip_nat_irc**" (d'autres modules sont disponibles):

```
[root@phoenix /]# modprobe iptable_nat
[root@phoenix /]# modprobe ip_nat_ftp
[root@phoenix /]# modprobe ip_nat_irc
```

- Bien entendu, tout comme nous initialisons la table "**Filter**" nous devons initialiser la tables **NAT** :

```
[root@phoenix /]# iptables -t nat -F
[root@phoenix /]# iptables -t nat -X
```

- Pour ce qui sont des cibles par défaut des chaînes de la table **NAT**, nous acceptons toutes les connexions. Il n'est pas nécessaire de faire pointer ces cibles sur "**DROP**", car la sécurité est établie au niveau de la table "**Filter**", par le "**DROP**" par défaut de la chaîne **FORWARD** :

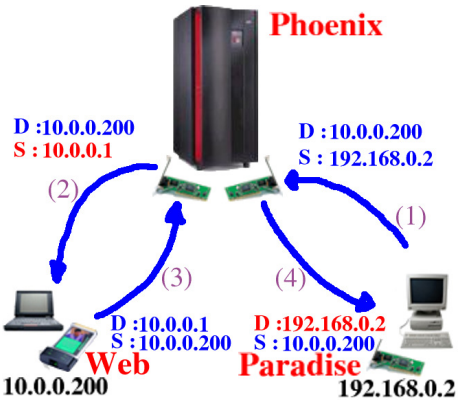
```
[root@phoenix /]# iptables -t filter -P FORWARD DROP
[root@phoenix /]# iptables -t nat -P PREROUTING ACCEPT
[root@phoenix /]# iptables -t nat -P POSTROUTING ACCEPT
[root@phoenix /]# iptables -t nat -P OUTPUT ACCEPT
```

- Bien, maintenant nous allons faire suivre sur le réseau **internet.net**, les connexions issues du réseau **local.net**. Bien entendu, nous ne ferons suivre que les connexions qui sont à destination du réseau **internet.net**, et non celles destinées à la machine Phoenix elle-même. Pour cela, nous allons avoir besoin d'utiliser la table "**Filter**" (he oui, encore !), afin de faire suivre les paquets venant de la carte **eth0** (phoenix0.local.net) à la carte **eth1** (phoenix1.internet.net), et vice versa. En plus de cela, comme ce sont uniquement les connexions initialisées par le réseau interne que nous désirons faire sortir, nous rajouterons un peu de suivi de connexion.

```
[root@phoenix ]# iptables -t filter -A FORWARD -i eth0 -o eth1 -s 192.168.0.0/24 \
-d 0.0.0.0/0 -m state --state ! INVALID -j ACCEPT
[root@phoenix ]# iptables -t filter -A FORWARD -i eth1 -o eth0 -s 0.0.0.0/0 \
-d 192.168.0.0/24 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

L'accumulation des paramètres "-i", "-o", "-s", "-d" et "--state" permettent de garantir que seul les connexions initialisées par le domaine local.net seront autorisées à passer, et non l'inverse.

- Au point où nous en somme, les paquets venant de local.net et sortant par l'interface eth1, ont :
  - ***pour adresse de destination***, une adresse du réseau **internet.net**, ce qui est parfaitement normale. Celle de web.internet.net par exemple
  - ***pour adresse source***, l'adresse d'une machine du domaine **local.net**, comme par exemple celle de paradise.local.net (voir lien "(1)"). Oui, mais ce domaine est un réseau privé, dont l'adresse IP n'est pas du tout routable sur Internet.
    Conclusion : lorsque web.internet.net recevra la requête, il ne saura absolument pas où envoyer la réponse... C'est assez ennuyeux, non ?



Il va donc falloir que Phoenix subtilise l'adresse IP source des trames qui le traverse, et la remplace par la sienne. C'est là qu'intervient la table **NAT**, avec la cible "**MASQUERADE**" (pour "masquage" en anglais) :

```
[root@phoenix /]# iptables -t nat -A POSTROUTING -o eth1 -s 192.168.0.0/24 -j MASQUERADE
```

Que deviennent les paquets revenant du réseau internet.net sur l'interface eth1 ? Ils ont pour adresse source **web.internet.net**, et pour adresse cible phoenix1.internet.net (voir lien "(3)"). Donc si ils sont

recopiés sur le réseau local.net , ils ne saurons pas qu'ils doivent aller sur **paradise.local.net**, non ? En théorie, ceci est exacte, et demanderait la mise en place d'une seconde règle de "**de**" "**MASQUERADE**". Mais en fait, c'est inutile, car avec la première règle, Netfilter gère une table en mémoire qui suit ces masquage d'adresses IP, et elle réassigne aux connexions rentrantes la bonnes adresse IP cibles.

En fait, ce serait même terriblement dangereux de mettre en place une telle règle de ce type. En effet, cela pourrait faire passer pour des connexions de Phoenix, des connexions initialisées par les machines de internet.net ! C'est comme si vous poussiez vous-même le loup dans la bergerie ! Donc l'utilisation de la table **NAT** et de la cible "**MASQUERADE**" doivent se faire avec le plus rigueur possible, souvenez vous en bien.

- Enfin, maintenant que tout le **NAT** est configuré, il ne reste plus qu'à autoriser **Linux** à faire du **routage** et à jouer son rôle de gateway. Pour cela, il faut utiliser simplement écrire un "**1**" dans le "**/proc/sys/net/ipv4/ip_forward**". Ce fichier est en fait un fichier virtuel, qui permet de dialoguer directement avec le **kernel**. Nous allons donc utiliser la commande suivante:

```
[root@phoenix /]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Le **NAT** étant maintenant configuré et activé, il reste à configurer les machines du réseau **local.net**, afin de leur indiquer quelle est la **passerelle**. C'est chose faite en lançant par exemple sur paradise.local.net :

```
[root@paradise /]# route add default gw phoenix0.local.net
[root@paradise /]# route
Table de routage IP du noyau
Destination   Passerelle      Genmask      Indic  Metric  Ref   Use  Iface
192.168.0.0       *        255.255.255.0    U       0      0     0    eth0
127.0.0.0         *          255.0.0.0      U       0      0     0     lo
default     phoenix0.sky.ne    0.0.0.0      UG      0      0     0    eth0
```

Bien, tout est en place. Voyons ce que cela donne. Depuis paradise.local.net, tentons une connexion sur web.internet.net:

```
[coco@paradise /]$ nmap web.internet.net -p 80,443

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on web.0.0.10.in-addr.arpa (10.0.0.200):
Port      State     Service
80/tcp    open       http
443/tcp   open       https

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

Bien ! Comme on le voit, la connexion se fait d'un réseau à l'autre, à travers Phoenix. Opération réussie donc ! Cela qui nous donne l'occasion de trouver ici le script **iptables** de cet exemple.

Remarques :

Dans tout nos exemples, nous avons systématiquement autorisé tout le réseau local.net à accéder à **internet.net**. Cependant, nous aurions très bien pu limiter cet accès à uniquement quelques machines de notre réseau interne. Pour cela, il suffit de changer les options "**-s**" et "**-d**" de nos règles de "**FORWARD**", et d'indiquer l'adresse IP de la machine qui est autorisée à passer d'un réseau à un autre.

Exemple :

```
[root@phoenix ]# iptables -t filter -A FORWARD -i eth0 -o eth1 -s 192.168.0.2 -d 0.0.0.0/0
-m state --state ! INVALID -j ACCEPT
[root@phoenix ]# iptables -t filter -A FORWARD -i eth1 -o eth0 -s 0.0.0.0/0 -d 192.168.0.2
-m state --state ESTABLISHED,RELATED -j ACCEPT
```

Et la sécurité ? pirate.internet.net peut-il accéder à paradise.local.net ? Techniquement oui, il suffit que pirate.internet.net déclare phoenix1.internet.net comme étant sa **passerelle** pour le réseau **internet.net** :

```
[root@pirate /]# route add default gw phoenix1.internet.net
[root@pirate /]# route
Table de routage IP du noyau
Destination  Passerelle   Genmask       Indic  Metric   Ref  Use  Iface
10.0.0.0         *       255.0.0.0       U       0       0    0   eth0
127.0.0.0        *       255.0.0.0       U       0       0    0    lo
default    phoenix1.intern 0.0.0.0       UG      0       0    0   eth0
```

Puis, de lancer par exemple un "**ping**" sur l'adresse IP de paradise.local.net :

```
[root@pirate /]# ping -c 3 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2014ms
```

Bien, notre machine est correctement sécurisée. Mais il ne faut pas grand chose pour que cette état idéal se transforme en un cauchemar...

En effet, regardons les traces qu'on laissé la dernière connexion de pirate.internet.net dans les statistiques de Netfilter :

```
[root@phoenix /]# iptables -L -v -n -t nat
Chain PREROUTING (policy ACCEPT 3 packets, 252 bytes)
Pkts    bytes  target   prot    opt    in    out      source        destination
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
Pkts    bytes  target   prot    opt    in    out      source        destination
0       0    MASQUERADE all     --     *    eth1 192.168.0.0/24     0.0.0.0/0
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
Pkts    bytes  target   prot    opt    in    out      source        destination
```

Cette commande nous indique que 3 paquets ont été acceptés par le comportement par défaut de la chaîne "**PREROUTING**" de la table "**NAT**". C'est tout à fait normal, ce sont nos 3 paquets de **ping** qui arrivent de **pirate.internet.net** et qui s'apprêtent à passer à travers Phoenix. Mais qui va les arrêter ?

```
[root@phoenix/]# iptables -L -v -n -t filter
Chain INPUT (policy DROP 0 packets, 0 bytes)
Pkts    bytes  target   prot    opt    in    out      source        destination
0       0      ACCEPT   all     --    lo     *      0.0.0.0/0       0.0.0.0/0
0       0      ACCEPT   all     --   eth0    *    192.168.0.0/24  192.168.0.0/24
Chain FORWARD (policy DROP 3 packets, 252 bytes)
pkts    bytes  target   prot    opt    in    out      source        destination
0       0      ACCEPT   all     --   eth0  eth1 192.168.0.0/24     0.0.0.0/0    state
       NEW,RELATED,ESTABLISHED
0       0      ACCEPT   all     --   eth1  eth0    0.0.0.0/0     192.168.0.0/24 state
       RELATED,ESTABLISHED
Chain OUTPUT (policy DROP 0 packets, 0 bytes)
pkts    bytes  target   prot    opt    in    out      source        destination
0       0      ACCEPT   all     --     *     lo     0.0.0.0/0       0.0.0.0/0
0       0      ACCEPT   all     --     *    eth0 192.168.0.0/24  192.168.0.0/24
```

Cette commande là nous indique que 3 paquets ont été détruits par le comportement par défaut de la chaîne "**FORWARD**" de la table "**Filter**". Là encore, c'est ce à quoi nous attendions. Pourquoi ? Parce qu'aucune des autres règles de la chaîne "FORWARD" de cette table ne les ont laissé passer.

Maintenant, changeons seulement le comportement par défaut de la table "**FORWARD**" de la table "**Filter**" :

```
[root@phoenix /]# iptables -t filter -P FORWARD ACCEPT
```

Et relançons notre "ping" depuis pirate.internet.net :

```
[intrus@pirate /]$ ping -c 3 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=254 time=0.957 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=254 time=0.987 ms
```

```
64 bytes from 192.168.0.2: icmp_seq=3 ttl=254 time=0.957 ms
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.957/0.967/0.987/0.014 ms
```

!? Notre intrus peut accéder sans problème à l'intérieur de notre réseau! Et tout cela à cause d'un malheureuse règle par défaut ? Regardons les statistiques d'iptables pour la table "**Filter**" :

```
[root@phoenix /]# iptables -L -v -n -t filter
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts    bytes  target   prot    opt    in   out      source       destination
0       0      ACCEPT   tcp     --     *    *     0.0.0.0/0      0.0.0.0/0    tcp  dpt:6000
0       0      ACCEPT   tcp     --     *    *     0.0.0.0/0      0.0.0.0/0    tcp  dpt:6000
0       0      ACCEPT   all     --    lo    *     0.0.0.0/0      0.0.0.0/0
0       0      ACCEPT   all     --   eth0   *   192.168.0.0/24 192.168.0.0/24
Chain FORWARD (policy ACCEPT 3 packets, 252 bytes)
pkts    bytes  target   prot    opt    in   out      source       destination
3 252 ACCEPT all -- eth0 eth1 192.168.0.0/24 0.0.0.0/0 state NEW,RELATED,ESTABLISHED
0 0 ACCEPT all -- eth1 eth0 0.0.0.0/0 192.168.0.0/24 state RELATED,ESTABLISHED
Chain OUTPUT (policy DROP 0 packets, 0 bytes)
pkts    bytes  target   prot    opt    in   out      source       destination
0       0      ACCEPT   tcp     --     *    *     0.0.0.0/0      0.0.0.0/0    tcp  spt:6000
0       0      ACCEPT   tcp     --     *    *     0.0.0.0/0      0.0.0.0/0    tcp  spt:6000
0       0      ACCEPT   all     --     *   lo     0.0.0.0/0      0.0.0.0/0
0       0      ACCEPT   all     --     *   eth0 192.168.0.0/24 192.168.0.0/24
```

Voilà quelque chose de très intéressant :

3 paquets sont passés par la cible "**ACCEPT**" par défaut de la chaîne "**Filter**" . Se sont les 3 pings venant de pirate.internet.net à destination de paradise.local.net. Notre trou de sécurité se trouve indéniablement ici.

3 autres paquets ont utilisés la règle basée sur du suivi de connexion , partant du réseau local.net et allant sur internet.net. Bien sûr, c'est tout à fait normal. Car pour Netfilter, une connexion local.net -> internet.net est bien initialisée ou en relation avec une autre...

Regardons maintenant les statistiques de la table "**NAT**" :

```
[root@phoenix /]# iptables -L -v -n -t nat
Chain PREROUTING (policy ACCEPT 3 packets, 252 bytes)
Pkts    bytes  target   prot    opt    in   out      source       destination
Chain POSTROUTING (policy ACCEPT 3 packets, 252 bytes)
Pkts    bytes  target   prot    opt    in   out      source       destination
0       0    MASQUERADE all     --     *   eth1 192.168.0.0/24    0.0.0.0/0
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
Pkts    bytes  target   prot    opt    in   out      source       destination
```

3 paquets sont passés par les chaînes "**PREROUTING**" puis "**POSTROUTING**" afin de sortir du réseau internet.net et atteindre le réseau **local.net**. Par contre, il n'y a pas de traces des paquets de réponse de paradise.local.net à pirate.internet.net. Ceci est dû au mécanisme interne de Netfilter qui identifie les connexions en réponse, et qui ne les fait pas passer par la table NAT. C'est pour cela, et comme vu plus haut, qu'il n'y a pas besoin d'écrire des règles spécifiques pour le "**MASQUERADING** rentrant".

De cette explication nous pourrons en tirer 3 leçons :

Il faut toujours initialiser les cibles par défaut , même pour les chaînes des tables que nous ne pensons peut-être pas utiliser. Il faut en toute situation connaître l'état de ces cibles. Un script Netfilter devrait donc toujours commencer par :

```
# Initialisation de la table FILTER
iptables -t filter -F
iptables -t filter -X
iptables -t filter -P INPUT DROP
iptables -t filter -P OUTPUT DROP
iptables -t filter -P FORWARD DROP
```

```
# Initialisation de la table NAT
iptables -t nat -F
iptables -t nat -X
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
iptables -t nat -P OUTPUT ACCEPT
# Initialisation de la table MANGLE
iptables -t mangle -F
iptables -t mangle -X
iptables -t mangle -P PREROUTING ACCEPT
iptables -t mangle -P INPUT ACCEPT
iptables -t mangle -P OUTPUT ACCEPT
iptables -t mangle -P FORWARD ACCEPT
iptables -t mangle -P POSTROUTING ACCEPT
```

Avant de mettre en place un système de **NAT**, il faut bien prendre en compte tous les paramètres, et se rendre compte de l'impact que cela peut avoir sur la sécurité de votre réseau. Il n'y a rien de pire que de copier/coller un mauvais morceau de script "**iptables**", et de se dire que cela doit bien marcher tout seul. Voir de bidouiller affreusement le script dans son coin, et de jouer à l'apprenti sorcier !

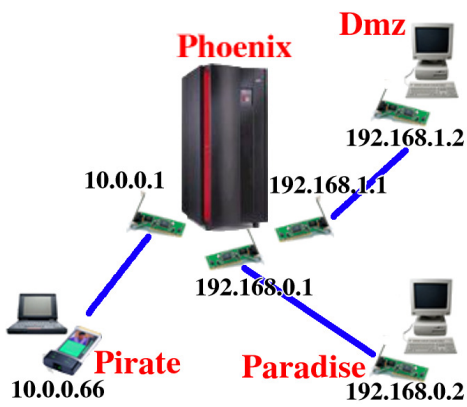
En cas de doute, ou de problème sur l'utilisation de Netfilter, il ne fait pas hésiter à utiliser la commande "**iptables -L -v -t [table]**" qui donne plein d'informations quand aux nombre de paquets passant par une règle ou un comportement par défaut.

Bien, l'IP **masquerading** étant vu, passons au port **forwarding**.

B. Port forwarding

Bien caché derrière notre firewall Phoenix, notre réseau local.net ne craint plus grand chose. De plus, avec ce que nous avons vu au chapitre précédent , les machines de notre réseau local peuvent rejoindre le réseau internet.net.

Maintenant, nous voudrions avoir un serveur (HTTP, FTP, IRC, peer-to-peer, etc...) qui puisse proposer des informations et des services non plus sur le réseau local.net, mais carrément sur le réseau internet.net. L'emplacement d'un tel service devrait se faire en toute logique sur la machine Phoenix, qui est le seul accès que nous ayons sur le réseau internet.net. Mais ceci serait une très mauvaise idée . Pourquoi ? Parce qu'un tel accès est inévitablement une faille dans notre système de sécurité. Grâce à ce service que nous proposons, l'intrus pourrait essayer de se l'accaparer, puis d'en profiter pour prendre la main sur Phoenix, et d'enfin de court-circuiter toutes nos règles Netfilter. Notre machine se trouverait alors sans aucune protection vis à vis de l'extérieur... Inquiétant, non ? Au minimum, il faudrait configurer ce service pour fonctionner sous un utilisateur n'ayant que peu de droits, et en plus dans un chroot. Mais même avec cela, la sécurité ne pourrait pas être garantie.



DMZ pour port fowarding

Alors, comment faire ? C'est là qu'intervient le suivit de port ("port fowarding" en anglais). L'idée d'un tel système est que toute connexion entrante sur un certain port de phoenix1.internet.net (par exemple le port 80, pour du HTTP), soit automatiquement redirigée vers une autre machine de local.net : paradise.local.net par exemple. Vu de l'extérieur, on aurait l'impression que le port 80 de phoenix1.internet.net serait ouvert, mais en fait, ce serait le port 80 de paradise.local.net qui serait réellement ouvert, et sur lequel tournerait un serveur Apache.

Est-ce réellement une "solution miracle" ? Presque, mais à condition de prendre quelques précautions :

Dans cette configuration, nous ne faisons que déplacer le problème de sécurité sur Paradise. Il faut donc que cette machine soit très bien sécurisée, avec les paramétrages vus dans la section "risque" du chapitre II. Mais ce ne sera pas suffisant. Paradise devenant donc un serveur web à part entière, et du fait des risques énoncés, Phoenix devra donc s'en méfier comme de la peste. Fini les gentilles règles Netfilter laissant paradise.local.net pleinement accéder à phoenix0.local.net. Il faudra durcir tout cela, et utiliser le conntrack ! Dans notre esprit, paradise.local.net devra devenir aussi soupçonnable que pirate.internet.net.

Enfin, dans le cas où un intrus prendrait le contrôle de Paradise, et même si Phoenix à des règles Netfilter très strictes, rien n'empêcherait l'intrus de s'attaquer à une autre machine du réseau local.net ! Voir dans le pire des cas, de s'approprier le contrôle d'une autre machine de local.net, puis d'attaquer Phoenix par l'arrière, en profitant d'éventuelles règles Netfilter plus souples vis à vis de ce 2nd hôte...

En terme de sécurité informatique, la paranoïa peut devenir un jeu d'esprit où il faut systématiquement envisager les pires cas, et les hypothèses les plus tordues. Dans le cas de notre réseau, les 3 problèmes ci-dessus peuvent êtres réglés par une technique plutôt efficace, appelée DMZ ("DeMilitarized Zone ou "Zone DéMilitarisée" en français). La mise en place d'une telle DMZ se fait en installant une 3ème carte réseau sur Phoenix ("eth2"), sur laquelle on ne reliera que une seule machine qui ne servira qu'au seul usage de serveur web.

On appellera cette machine DMZ, tout simplement. Voir à ce sujet l'illustration ci-contre. Mais ceci sort du cadre de ce document, et d'un réseau personnel, je n'en parlerai donc pas plus longuement.

Mais revenons à un réseau un peu plus simple, et baissons d'un cran notre paranoïa. Nous allons supposer que la machine qui hébergera notre serveur HTTP est fiable, et que personne ne peut en prendre la main (ceci n'est qu'une hypothèse bien sûr !). Nous la laisserons donc dans le réseau local.net, en compagnie des autres machines de notre réseau interne.

Que devons nous faire ?

Tout d'abord, nous devons initialiser la table NAT, tout comme nous l'avons fait pour le IP masquerading. Et aussi bien sûr charger le module "iptables_nat" :

```
[root@phoenix /]# modprobe iptable_nat
[root@phoenix /]# iptables -t nat -F
[root@phoenix /]# iptables -t nat -X
[root@phoenix /]# iptables -t filter -P FORWARD DROP
[root@phoenix /]# iptables -t nat -P PREROUTING ACCEPT
[root@phoenix /]# iptables -t nat -P POSTROUTING ACCEPT
[root@phoenix /]# iptables -t nat -P OUTPUT ACCEPT
```

Comme nous voulons partager un serveur HTTP, il peut être intéressant d'autoriser les machines de internet.net de pouvoir utiliser le "ping" sur notre serveur. Ce n'est absolument pas une obligation, mais cela peut-être pratique pour les Internautes qui, avant de lancer la connexion HTTP, vérifient que notre machine est bien active. Ce choix relève plus d'une bonne manière qu'autre chose, et n'est donc pas obligatoire. Notre "port forwarding" pourra très bien marcher sans cela. On utilise quand même du "suivi de connexion", afin de ne pas accepter les paquets ICMP volontairement mal formatés :

```
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -s 0.0.0.0/0 -d 10.0.0.1 -p icmp \
-m state --state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -t filter -A OUTPUT -o eth1 -s 10.0.0.1 -d 0.0.0.0/0 -p icmp \
-m state --state RELATED,ESTABLISHED -j ACCEPT
```

Au passage, vous noterez qu'il n'est pas nécessaire d'ouvrir d'autres accès. Et notamment via les chaînes "INPUT" et "OUTPUT" afin autoriser l'échange avec le port 80. Pourquoi ? Simplement parce que dans le cas du port forwarding, les trames à destination de paradise.local.net ne sont pas destinées aux processus locaux, et donc elle ne passent pas à travers les chaînes "INPUT" et "OUTPUT". En fait, comme on le verra tout de suite, seul la chaîne "FORWARD" de la table "Filter" devra être modifiée.

Nous allons maintenant faire suivre un certain type de paquets de l'interface "eth1" à l'interface "eth0" : Ce sera les paquets à destination du serveur HTTP. De même que nous laisserons passer les paquets en réponse. Là encore, nous utilisons du "conntrack" afin d'améliorer la sécurité de ce suivi de port :

```
[root@phoenix /]# iptables -t filter -A FORWARD -i eth1 -o eth0 -s 0.0.0.0/0 \
-d 192.168.0.2 -p tcp --dport 80 -m state --state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -t filter -A FORWARD -i eth0 -o eth1 -s 192.168.0.2 \
-d 0.0.0.0/0 -p tcp --sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Nous arrivons à la 1er spécificité du "port forwarding". Pour les paquets entrant sur phoenix1.internet.net et à destination du port 80, nous allons modifier l'adresse de destination. Ce ne sera plus phoenix1.internet.net, mais paradise.local.net, c'est à dire notre réel serveur HTTP. Pour cela, nous allons utiliser dans la table NAT, la chaîne "PREROUTING" et la cible "DNAT" (pour "Destination Network Adress Translation" en anglais).

```
[root@phoenix /]# iptables -t nat -A PREROUTING -i eth1 -s 0.0.0.0/0 -d 10.0.0.1 -p tcp \
--dport 80 -m state --state ! INVALID -j DNAT --to-destination 192.168.0.2:80
```

Au passage, on voit que l'on peut aussi modifier le port de destination, et mettre autre chose que 80. Cela peut-être utile si le serveur HTTP sur paradise.local.net tourne sur autre chose que le port 80.

La seconde spécificité du "**port forwarding**", c'est de modifier aussi l'adresse source de la requête. En effet, si nous laissons l'adresse source actuelle (une adresse de 10.0.0.0/8 par exemple), paradise.local.net sera bien ennuyé pour répondre, car ce sera une adresse d'un réseau qu'il ne peut pas joindre. Évidement, nous pourrions indiquer à paradise.local.net que phoenix0.local.net est une passerelle, et dans ce cas là les paquets retrouveraient tout de suite la sortie du réseau local.net. Mais, et même si cette solution marche effectivement très bien, c'est une solution particulièrement mal propre, comme nous le verrons un peu plus loin. Donc, modifions cette adresse source :

```
[root@phoenix /]# iptables -t nat -A POSTROUTING -o eth0 -s 0.0.0.0/0 \
-d 192.168.0.2 -p tcp --dport 80 \
-m state --state ! INVALID -j SNAT --to-source 192.168.0.1
```

Bien, au niveau de Netfilter, tout est configuré. Il nous reste qu'à activer le NAT par la commande que nous connaissons déjà :

```
[root@phoenix /]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Voici le **script iptables** décrit ici.

```
#!/bin/sh
#
#################################################################################
# NOM: iptable-portforwarding-1.sh
#
# COMMENTAIRE : Exemple d'utilisation du port fowarding
#               pirate.internet.net.sky.net va pouvoir utiliser le HTTP de
#               paradise.sky.net
#################################################################################

# Suppression de toutes les chaînes pré-définies de la table FILTER
iptables -t filter -F

# Suppression de toutes les chaînes utilisateur de la table FILTER
iptables -t filter -X

# Par defaut, toute les paquets de la table FILTER sont détruits
iptables -t filter -P INPUT DROP
iptables -t filter -P OUTPUT DROP
iptables -t filter -P FORWARD DROP

# Autorise l'interface loopback à dialoguer avec elle-même
iptables -t filter -A OUTPUT -o lo -j ACCEPT
iptables -t filter -A INPUT  -i lo -j ACCEPT

# Autorise les connexions avec le réseau 192.168.0.0/24 connecté à l'interface eth0
iptables -t filter -A OUTPUT -o eth0 -d 192.168.0.0/24 -j ACCEPT
iptables -t filter -A INPUT  -i eth0 -s 192.168.0.0/24 -j ACCEPT


#################################################################################
# Port fowarding
#################################################################################

# Chargement des modules du NAT
modprobe iptable_nat    ; # Module principal du NAT

# Suppression de toutes les chaînes pré-définies de la table NAT
```

```
iptables -t nat -F

# Suppression de toutes les chaînes utilisateur de la table NAT
iptables -t nat -X

# Par defaut, toute les paquets de la table NAT sont DROP
iptables -t nat -P PREROUTING  ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
iptables -t nat -P OUTPUT      ACCEPT

# Autorise phoenix1.internet.net à répondre aux ping des machines de internet.net
iptables -t filter -A INPUT  -i eth1 -s 0.0.0.0/0 -d 10.0.0.1  -p icmp -m state \
--state ! INVALID -j ACCEPT
iptables -t filter -A OUTPUT -o eth1 -s 10.0.0.1  -d 0.0.0.0/0 -p icmp -m state \
--state RELATED,ESTABLISHED -j ACCEPT

# Fait suivre les trames venant d'Internet et à destination de paradise.sky.net
iptables -t filter -A FORWARD -i eth1 -o eth0 -s 0.0.0.0/0   -d 192.168.0.2 -p tcp \
--dport 80 -m state --state ! INVALID -j ACCEPT

# Fait suivre les trames venant paradise.sky.net et à destination d'Internet
iptables -t filter -A FORWARD -i eth0 -o eth1 -s 192.168.0.2 -d 0.0.0.0/0   -p tcp \
--sport 80 -m state --state RELATED,ESTABLISHED -j ACCEPT

# Pour les trames arrivant sur le port 80 de phoenix1.internet.net,
# change l'adresse de destination en paradise.sky.net
iptables -t nat -A PREROUTING  -i eth1 -s 0.0.0.0/0   -d 10.0.0.1  -p tcp --dport 80 \
 -m state --state ! INVALID -j DNAT --to-destination 192.168.0.2:80

# Pour les trames arrivant sur le port 80 de phoenix1.internet.net,
# change l'adresse source en phoenix0.sky.net
iptables -t nat -A POSTROUTING -o eth0 -s 0.0.0.0/0 -d 192.168.0.2 -p tcp --dport 80 \
-m state --state ! INVALID -j SNAT --to-source      192.168.0.1

# Pour le debugage:
# Log les trames utilisant les cibles par defaut de PREROUTING et de POSTROUTING.
# Cela peut aider à trouver une mauvaise configuration de ces chaînes
#iptables -t nat -A PREROUTING  -j LOG --log-prefix PreRoutingError
#iptables -t nat -A POSTROUTING -j LOG --log-prefix PostRoutingError

# Activation du ROUTAGE dans le kernel
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Il ne nous reste plus qu'à vérifier que pirate.internet.net a bien accès à phoenix1.internet.net :

```
[intrus@pirate /]$ telnet phoenix1.internet.net 80
Escape character is '^]'.
GET /
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Mozilla/4.61 [en] (X11; I; Linux 2.2.9-23mdk i686)
[Netscape]">
<meta name="Author" content="MandrakeSoft">
<title>Welcome to the Advanced Extranet Server, ADVX!</title>
<LINK REL="SHORTCUT ICON" HREF="/favicon.ico">
<!-- Background white, links blue (unvisited), navy (visited), red (active) --> </head>
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#000080" alink="#FF0000">
<table border=0>
<tr>
<td valign=top><a href=http://www.advx.org><img border=0 src=/icons/advx.png
ALT="Powered by ADVX.org software" height=47 width=102 align=LEFT></a></td>
<td valign=top width=100%><h2><center>Welcome to paradise.local.net
</center></h2>
</td></tr>
```

La fin de la réponse a été tronquée. On voit bien que, malgré que l'on accède à phoenix1.internet.net, c'est paradise.local.net qui répond. Regardons maintenant le log du serveur Apache de paradise.local.net :

```
[root@paradise /]# tail -1 /var/log/httpd/access_log
192.168.0.1 - - [06/Jul/2003:21:02:21 +0200] "GET /" 200 6988 "-" "-"
```

On voit la connexion faite par pirate.internet.net, et le téléchargement de la page web. Cependant, on notera que l'adresse IP qui est logée n'est pas celle de pirate.internet.net, mais bien celle de phoenix1.internet.net, ce qui est tout à fait normal, du fait de l'utilisation de la cible "SNAT".

Enfin, regardons ce que nous donne les statistiques de Netfilter sur Phoenix :

```
[root@phoenix /]# iptables -L -v -n -t filter
Chain INPUT (policy DROP 0 packets, 0 bytes)
Pkts  bytes target  prot   opt   in   out     source      destination
0     0     ACCEPT  tcp    --    *    *     0.0.0.0/0     0.0.0.0/0    tcp dpt:6000
0     0     ACCEPT  tcp    --    *    *     0.0.0.0/0     0.0.0.0/0    tcp dpt:6000
0     0     ACCEPT  all    --   lo    *     0.0.0.0/0     0.0.0.0/0
0     0     ACCEPT  all    --  eth0   *  192.168.0.0/24   0.0.0.0/0
0     0     ACCEPT  icmp   --  eth1   *     0.0.0.0/0      10.0.0.1     state
NEW,RELATED,ESTABLISHED
Chain FORWARD (policy DROP 0 packets, 0 bytes)
Pkts  bytes target  prot   opt   in   out     source      destination
8     430   ACCEPT  tcp    --  eth1 eth0   0.0.0.0/0     192.168.0.2   tcp dpt:80 state
NEW,RELATED,ESTABLISHED
8     7412  ACCEPT  tcp    --  eth0 eth1  192.168.0.2     0.0.0.0/0    tcp spt:80 state
RELATED,ESTABLISHED
Chain OUTPUT (policy DROP 0 packets, 0 bytes)
pkts  bytes target  prot   opt   in   out     source      destination
0     0     ACCEPT  tcp    --    *    *     0.0.0.0/0     0.0.0.0/0    tcp spt:6000
0     0     ACCEPT  tcp    --    *    *     0.0.0.0/0     0.0.0.0/0    tcp spt:6000
0     0     ACCEPT  all    --    *   lo     0.0.0.0/0     0.0.0.0/0
0     0     ACCEPT  all    --    *  eth0    0.0.0.0/0   192.168.0.0/24
0     0     ACCEPT  icmp   --    *  eth1     10.0.0.1     0.0.0.0/0    state RELATED,
ESTABLISHED
[root@phoenix /]# iptables -L -v -n -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts  bytes target  prot   opt   in   out     source      destination
1     60     DNAT   tcp    --  eth1   *     0.0.0.0/0      10.0.0.1     tcp dpt:80 state
NEW,RELATED,ESTABLISHED to:192.168.0.2:80
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts  bytes target  prot   opt   in   out     source      destination
1     60     SNAT   tcp    --    *  eth0    0.0.0.0/0     192.168.0.2   tcp dpt:80 state
NEW,RELATED,ESTABLISHED to:192.168.0.1
```

La requête et l'envoi de la page HTML ont demandés au total 2 fois 8 paquets, qui sont passés à travers les chaînes "FORWARD".

On voit bien que les paquets entrant dans phoenix1.internet.net sont bien passés par les cibles "DNAT" et "SNAT" des chaînes "PREROUTING" et "POSTROUTING". Par contre ces statistiques ne montre pas les paquets en réponses à la demande de connexion. C'est dû à la même routine de suivi de connexion de Netfilter que pour l'IP masquerading.

Nous pourrions nous arrêter ici pour le port fowarding, mais un dernier point est intéressant à soulever : que se passe t'il si nous n'incluons pas la règles SNAT, ce qui est indiqué plus haut comme étant une " méthode sale " ?

Commençons par retirer cette règle :

```
[root@phoenix /]# iptables -D POSTROUTING 1 -t nat
```

Puis, indiquons à Paradise que sa passerelle est phoenix0.local.net:

```
[root@paradise /]# route add default gw phoenix0.local.net
[root@paradise /]# route
```

```
Table de routage IP du noyau
Destination   Passerelle     Genmask      Indic  Metric   Ref Use  Iface
192.168.0.0       *       255.255.255.0    U       0       0   0    eth0
127.0.0.0         *         255.0.0.0      U       0       0   0     lo
default     phoenix0.sky.ne  0.0.0.0        UG      0       0   0    eth0
```

Et maintenant, demandons à nouveau notre page HTML depuis pirate.internet.net. Suite à cette connexion, les statistiques d'"iptables" nous donne :

```
[root@phoenix /]# iptables -L -v -n -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
1 60 DNAT tcp -- eth1 * 0.0.0.0/0 10.0.0.1 tcp dpt:80 state NEW,RELATED, ESTABLISHED
to:192.168.0.2:80
Chain POSTROUTING (policy ACCEPT 1 packets, 60 bytes)
pkts bytes target prot opt in out source destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
```

Comme précédemment, un paquet est passé par le chaîne "PREROUTING", mais aucun n'est passé par de règles "POSTROUTING". Et pour cause, il n'y en a plus. Par contre, le paquet de retour est passé par la règle par défaut de la chaîne POSTROUTING... C'est parce que les paquets n'empruntent pas les mêmes canaux en entrée et en sortie que j'estime cette méthode comme étant "sale".

Enfin ce qui est le plus intéressant c'est le log d'Apache sur paradise.local.net :

```
[root@paradise /]# tail -1 /var/log/httpd/access_log
10.0.0.66 - - [06/Jul/2003:22:27:43 +0200] "GET /" 200 6988 "-" "-"
```

Cette fois ci, c'est bien l'adresse IP de la machine à distance qui est stockée, forcément, puisque Phoenix ne la dissimule plus.

Vous trouverez le script de cet exemple ici, mais comme indiqué, plus haut c'est la méthode "sale" pour effectuer du port forwarding...

Ceci conclue donc l'utilisation du port forwarding. Dans le cadre d'un usage personnel, cette technique est assez peu utile : qui chez lui héberge son propre serveur HTTP, FTP, IRC, etc...

Pas utile donc ? En fait non, il y a bien un usage personnel fort utile où l'on peut utiliser le port forwarding, et qui est particulièrement à la mode au jour où j'écris ces lignes : il s'agit du peer-to-peer (P2P). C'est un système d'échange de fichiers à travers Internet, où toutes les machines connectées partagent des fichiers sur un morceau de son disque dur. Chaque machine agit donc à la fois comme un client et un serveur. Une grand partie des clients pour ce type d'échange se trouvant sous Windows®, et comme c'est un OS dont l'usage courant est peu orienté sur la sécurité, on peut utiliser le port forwarding de Linux pour le protéger.

Avec les explications ci-dessus, vous pouvez sans difficulté écrire vos propres scripts iptables afin de renvoyer vos connexions entrantes vers la machine sur lequel tourne votre client P2P. Cependant, souvenez vous que vous ne faites que déplacer le problème de sécurité sur cette machine ci, et donc que c'est à vous d'en assurer la responsabilité. Si cette machine est par exemple sous Windows®, la présence de Netfilter en tête de votre réseau ne vous dispense pas de mettre en place un firewall logiciel sur cette machine, afin par exemple de surveiller ce que fait votre Windows®. Ceci étant, je me lave les mains de ce que vous pourriez faire de l'usage du port forwarding et du P2P. Et si je ne fournis aucun script d'exemple, c'est entre autre parce que je n'utilise pas ce type de logiciel...

Enfin, je dirais que la mise en place du port forwarding n'est pas forcément ce qui se fait le plus facilement, bien qu'en fait les règles ne soit pas beaucoup plus complexes que pour l'IP masquerading. Sur Internet, vous trouverez beaucoup de bribes d'explications sur cette techniques, et beaucoup de scripts plus où moins bien fait à ce sujet. Méfiez vous donc de ce que vous trouverez. Le mieux est sans aucun doute de tester votre configuration par étape, et de regarder systématiquement les statistiques d'iptables afin de voir par où passent vos paquets. D'une manière générale, si les paquets utilisent les cibles par défaut de PREROUTING ou de POSTROUTING, c'est qu'il y a probablement une mauvaise configuration de votre script. Enfin, comme indiqué dans les 2 scripts "iptables-portforwarding-*.sh" que vous pouvez télécharger, vous avez la possibilité d'utiliser la cible LOG afin d'analyser quelles sont les paquets qui utilisent ces cibles par défaut.

VI. Log (LOG / ULOG)

Jusqu'à présent, nous avons vu comment configurer nos règles avec Netfilter : définir les cibles par défaut, et supprimer des paquets. Mais à force de tout supprimer, il nous est difficile de connaître l'efficacité de notre firewall, et si il est intéressant de rajouter ou de supprimer des règles. D'où l'intérêt de logger certaines informations.

Dans ce qui suis, nous allons utiliser l'affreux anglicisme "logger" qui décrit l'action de stocker dans un "log" une certaine information.

A. LOG

C'est la méthode la plus standard pour logger des trames. Il s'agit simplement de créer une règle "iptables" dont la cible ("-j [cible]") n'est pas "ACCEPT" ou "DROP", mais tout simplement "LOG". Si nous reprenons le script "iptables-conntrack-1.sh", nous pouvons par exemple rajouter une dernière règle qui va logger tout ce qui na pas été accepté par la chaîne "INPUT". C'est très pratique, car ainsi nous serons averti de tout ce qui tente d'accéder aux processus utilisateurs de notre système. Pour cela, nous rajoutons cette règle en temps que dernière règle de la chaîne "INPUT" :

```
[root@phoenix /]# iptables -t filter -A INPUT -j LOG
```

Comme Netfilter est un élément du Kernel, ses logs sont donc des logs de ce dernier. Et comme tout bon log Kernel, ils se retrouve dans le fichier "/var/log/messages". Ainsi, si pirate.internet.net fait un "nmap" sur les ports HTTP et HTTPS de Phoenix, nous aurons :

```
[intrus@pirate /]# nmap phoenix1.internet.net -p 80,443
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 30 seconds
[root@phoenix /]# tail -14 /var/log/messages
Jul 8 23:04:59 phoenix nmbd[2178]: [2003/07/08 23:04:59, 0]
nmbd/nmbd_packets.c:send_netbios_packet(172)
Jul 8 23:04:59 phoenix nmbd[2178]: send_netbios_packet: send_packet() to IP 10.255.255.255
port 137 failed
Jul 8 23:04:59 phoenix nmbd[2178]: [2003/07/08 23:04:59, 0]
nmbd/nmbd_namequery.c:query_name(256)
Jul 8 23:04:59 phoenix nmbd[2178]: query_name: Failed to send packet trying to query name
WORKGROUP
Jul 8 23:07:03 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=28 TOS=0x00
PREC=0x00 TTL=38 ID=41593 PROTO=ICMP TYPE=8 CODE=0 ID=58020 SEQ=0
Jul 8 23:07:03 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=40 TOS=0x00
PREC=0x00 TTL=39 ID=51459 PROTO=TCP SPT=53120 DPT=80 WINDOW=4096 RES=0x00 ACK URGP=0
Jul 8 23:07:09 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=28 TOS=0x00
PREC=0x00 TTL=38 ID=2581 PROTO=ICMP TYPE=8 CODE=0 ID=58020 SEQ=256
Jul 8 23:07:09 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=40 TOS=0x00
PREC=0x00 TTL=39 ID=27444 PROTO=TCP SPT=53121 DPT=80 WINDOW=4096 RES=0x00 ACK URGP=0
Jul 8 23:07:15 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=28 TOS=0x00
PREC=0x00 TTL=38 ID=29887 PROTO=ICMP TYPE=8 CODE=0 ID=58020 SEQ=512
Jul 8 23:07:15 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=40 TOS=0x00
PREC=0x00 TTL=39 ID=48409 PROTO=TCP SPT=53122 DPT=80 WINDOW=4096 RES=0x00 ACK URGP=0
Jul 8 23:07:21 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=28 TOS=0x00
PREC=0x00 TTL=38 ID=37813 PROTO=ICMP TYPE=8 CODE=0 ID=58020 SEQ=768
Jul 8 23:07:21 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=40 TOS=0x00
PREC=0x00 TTL=39 ID=17449 PROTO=TCP SPT=53123 DPT=80 WINDOW=4096 RES=0x00 ACK URGP=0
Jul 8 23:07:27 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=28 TOS=0x00
PREC=0x00 TTL=38 ID=57294 PROTO=ICMP TYPE=8 CODE=0 ID=58020 SEQ=1024
Jul 8 23:07:27 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 LEN=40 TOS=0x00
PREC=0x00 TTL=39 ID=64622 PROTO=TCP SPT=53124 DPT=80 WINDOW=4096 RES=0x00 ACK URGP=0
```

Grâce au log, on voit que phoenix1.internet.net reçoit une série de commandes "ping" et de requêtes sur son port 80 de la part de pirate.internet.net.

Évidement, comme Phoenix ne répond pas à ces requêtes, Pirate fait plusieurs essais, et finalement ne testera même pas le port 443.

Un moyen pratique de suivre ses logs en temps réel est la commande, lancé en temps que root : " tail -f /var/log/messages". Cela affichera en permanence la fin de ce fichier de log. Pour l'arrêter, il suffit d'appuyer sur CTRL+C.

Mais ce fichier sert aussi (surtout !) à stocker tout les messages d'informations ou d'erreurs du Kernel. Il nous faut donc un moyen de retrouver ces messages de log Netfilter parmi tout les autres messages. On peut configurer avec le paramètre " --log-prefix=[Message de log]" la règle de log afin qu'elle rajoute systématiquement des messages en début de log. Ainsi :

```
[root@phoenix /]# iptables -t filter -A INPUT -s 10.0.0.66 -j LOG \
--log-prefix="AttackPirate"
[root@phoenix /]# iptables -t filter -A INPUT -s 10.0.0.200 -j LOG --log-prefix="AttackWeb"
```

indiquera clairement les connexions faites par les machines pirate.internet.net et web.internet.net :

```
[intrus@pirate /]$ nmap -p 80 phoenix1.internet.net
[intrus@web /]$ nmap -p 80 phoenix1.internet.net
[root@phoenix /]# tail -f /var/log/messages
Jul 8 23:26:06 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:26:06 phoenix kernel: AttackWeb IN=eth1 OUT= SRC=10.0.0.200 DST=10.0.0.1 ...
Jul 8 23:26:12 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:26:12 phoenix kernel: AttackWeb IN=eth1 OUT= SRC=10.0.0.200 DST=10.0.0.1 ...
Jul 8 23:26:12 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:26:12 phoenix kernel: AttackWeb IN=eth1 OUT= SRC=10.0.0.200 DST=10.0.0.1 ...
Jul 8 23:26:25 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:26:28 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:26:34 phoenix kernel: AttackPirate IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
```

Cependant, cette méthode de log a l'inconvénient que même avec des "--log-prefix", il reste difficile de séparer les messages de log de Netfilter des autres messages du Kernel. Comme en témoigne d'ailleurs le premier affichage du /var/log/message , où des messages de Samba (en début de log) sont mélangés aux messages de Netfilter.

Une autre méthode est de définir un "niveau de log" ("log level" en anglais), qui rajoute une autre information aux log, ce qui permet au demon de log (un programme appelé "syslogd") de stocker ces log dans un autre fichier. Par exemple, sur une Mandrake, on peut utiliser (voir "man syslogd", "man syslog.conf", "less /usr/include/sys/syslog.h" pour avoir plus d'informations sur ces commandes) :

```
[root@phoenix /]# iptables -t filter -A INPUT -j LOG --log-level=4
[root@phoenix /]# less /usr/include/sys/syslog.h
#define LOG_WARNING 4 /* warning conditions */
[root@phoenix /]# less /etc/syslog.conf
kern.=warn -/var/log/kernel/warnings
[intrus@pirate /]$ nmap -p 80 phoenix1.internet.net
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 60 seconds
[root@phoenix /]# tail -10 /var/log/kernel/warnings
Jul 8 18:55:43 phoenix kernel: MSDOS FS: Using codepage 850
Jul 8 19:26:34 phoenix kernel: i2c-amd756.o version 2.7.0 (20021208)
Jul 8 19:26:34 phoenix kernel: i2c-amd756.o: AMD768 bus detected and initialized
Jul 8 19:28:37 phoenix kernel: UDF-fs: No VRS found
Jul 8 23:53:58 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:54:01 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:54:07 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:54:10 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:54:13 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
Jul 8 23:54:19 phoenix kernel: IN=eth1 OUT= SRC=10.0.0.66 DST=10.0.0.1 ...
```

Comme on le voit, malgré que les log de Netfilter soient isolés dans un fichiers à part (/var/log/warnings), ce fichier n'en est pas moins partagé avec d'autres modules du Kernel, qui utilisent eux aussi le log Kernel de

niveau 4 <=> Warning. Dans le précédent log, on voit notamment des logs de "MSDOS FS" et "i2c-amd756.0" qui n'ont rien à voir avec Netfilter.

On pourrait s'imaginer utiliser une autre entrée du syslog, afin de vraiment séparer ces log. Le seul problème, c'est que les log de Netfilter sont des log du Kernel, et qu'à ce titre, on sera toujours obligé de les déclarer en temps que " kern." dans le "/etc/syslog.conf".

Alors ? Stocker ces logs à part est vraiment une chose impossible sous Linux ? Non, heureusement que non ! La solution s'appelle ULOG, et nous allons la voir tout de suite.

B. ULOG

ULOG est module du Kernel dont le développement a été fait par http://www.gnumonks.org/projects/. Il a été spécialement conçu pour recevoir les log de Netfilter. Il y a certaines (petites) contraintes à son utilisation :

Kernel récent : il faut un Kernel >= 2.4.18-pre8 pour pouvoir utiliser ce module.

Option de compilation : il faut que votre kernel soit compilé avec l'option CONFIG_IP_NF_TARGET_ULOG=m.

Une fois compilé, le module ipt_ULOG.o doit se trouver sur votre disque dur (/lib/modules/[version du kernel]/kernel/net/ipv4/netfilter/ipt_ULOG.o.gz par exemple) :

```
[root@phoenix /]# find /lib/modules/`uname -r`/ -iname "*ULOG*"
/lib/modules/2.4.21-0.13mdksmp/kernel/net/ipv4/netfilter/ipt_ULOG.o.gz
```

Sur une distribution "Mandrake 9.1", tout ceci est fait par défaut.

Vous devez récupérer et installer le demon "ulogd" sur votre machine. Personnellement, j'ai téléchargé la version 1.0, et je l'ai compilé et installé dans mon "/usr/local/sbin/ulogd".

Il faut configurer le demon ulogd. Pour cela, il faut éditer son fichier de configuration. Vous pouvez utiliser le mien. Les 2 informations importantes sont de spécifier qu'ULOG doit stocker ses logs sous forme de fichier texte, dans "/var/log/ulogd.syslogemu" par exemple.

Le demon ULOG ("ulogd") doit tourner sur votre machine. L'idéal est de le lancer au démarrage, en même temps que les autres demons de votre machine. A toute fin utile, voici le script de démarrage du demon ULOG que j'ai écris. Il se place dans le "/etc/init.d/". Pour le démarrer systématiquement, il faut créer un lien symbolique du "/etc/rc.d/rc3.d/S[un nombre]ulogd" ou du "/etc/rc.d/rc5.d/S[un nombre]ulogd" vers "/etc/init.d/ulog".

Exemple :

```
[root@phoenix scripts]# ls -la /etc/init.d/ulogd
-rwxr--r-- 1 root root 1821 jui 9 00:30 /etc/init.d/ulogd
[root@phoenix scripts]# ls -la /etc/rc.d/rc5.d/S10ulogd
lrwxrwxrwx 1 root root 15 mai 4 21:02 /etc/rc.d/rc5.d/S10ulogd -> ../init.d/ulogd
```

Une fois que tout ceci est mis en place :

Démarrez le demon ulog . Par exemple :

```
[root@phoenix /]# /etc/rc.d/rc5.d/S10ulogd start
```

Vérifiez que le démon fonctionne correctement :

```
[root@phoenix /]# cat /var/log/ulogd.log
Wed Jul 9 00:50:27 2003 <3> ulogd.c:474 ulogd Version 1.00 starting
Wed Jul 9 00:50:27 2003 <5> ulogd.c:688 initialization finished, entering main loop
```

Configurez Netfilter pour qu'il utilise la cible ULOG au lieu de la cible LOG. Ici, on log tout ce qui va être supprimé par la chaîne "INPUT" :

```
[root@phoenix /]# iptables -t filter -A INPUT -p all -j ULOG --ulog-prefix=DefaultDrop
```

Et attendez qu'un intrus vienne se frotter à votre Netfiler :

```
[root@phoenix /]# tail -f /var/log/ulogd.syslogemu
Jul 8 20:24:22 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32796 LEN=41
Jul 8 20:24:22 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=40 TOS=00 PREC=0x00 TTL=251 ID=0 DF PROTO=TCP SPT=110 DPT=33108 SEQ=0 ACK=1355878989
WINDOW=0 ACK RST URGP=0
Jul 8 20:24:32 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32797 LEN=41
Jul 8 20:24:54 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32799 LEN=41
Jul 8 20:25:04 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32800 LEN=41
Jul 8 20:25:22 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=78 TOS=00 PREC=0x00 TTL=104 ID=36090 PROTO=UDP SPT=1034 DPT=137 LEN=58
Jul 8 20:25:28 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=40 TOS=00 PREC=0x00 TTL=251 ID=0 DF PROTO=TCP SPT=110 DPT=33220 SEQ=0 ACK=1428948021
WINDOW=0 ACK RST URGP=0
Jul 8 20:25:34 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32802 LEN=41
Jul 8 20:25:38 phoenix DefaultDrop IN=ppp0 OUT= MAC= SRC=xx.xx.xx.xx DST=62.147.74.21
LEN=61 TOS=00 PREC=0x00 TTL=60 ID=0 DF PROTO=UDP SPT=53 DPT=32803 LEN=41
```

Ceci n'est qu'une simple portion de mon log de Netfilter, lors d'une connexion en RTC le 8 Juillet 2003. Remarquez le temps plutôt faible entre les accès. Il s'agit en fait de plusieurs "port scan" effectués par plusieurs machines... Et encore, mon Netfilter est configuré pour ne pas afficher les logs des requêtes P2P qui, bien que je n'ai aucun client P2P, inondent régulièrement ma machine...

Tout comme la cible LOG, la cible ULOG a des options intéressantes :

--ulog-prefix : préfixe des log. Même chose que pour "--log-prefix" de la cible LOG

--ulog-nlgroup : similaire à "--log-level" de la cible LOG

Conclusion : si vous voulez avoir des logs bien exploitables de votre Netfilter, utilisez ULOG plutôt que LOG. Ce n'est finalement pas si compliqué à installer, et c'est très pratique. Enfin, pour les maniaques de la sécurité qui ont plein de place disque et du CPU à revendre, vous pouvez spécifier au demon "ulogd" de stocker les paquets non pas dans un fichier, mais dans une base de donnés MySQL ou PostgreSQL.

VII. Autres astuces

A. Règles par défaut ("Policy")

Jusqu'à présent, nous avons vu que nous devions initialiser et définir les règles par défaut de ta table "Filter", et parfois celles de la table "NAT". En fait, ce n'est pas tout à fait la bonne manière de faire. A supposer par exemple que vous voulez faire uniquement du filtrage (option "-t filter"), vous pouvez très bien avoir laissé quelques règles de PREROUTING ou de POSTROUTING activées dans la table NAT. Dans ce cas, vous ne savez pas forcément ce que fait Netfilter, et il se peut qu'avec de telles configurations, il laisse passez des trames sans que vous ne vous en doutiez.

Donc il est est primordial que la première chose que vous fassiez dans un script Netfilter, c'est d'initialiser toutes les tables ("Filter", "NAT" et "Mangle") :

```
[root@phoenix /]# iptables -t filter -F
[root@phoenix /]# iptables -t filter -X
[root@phoenix /]# iptables -t filter -P INPUT DROP
[root@phoenix /]# iptables -t filter -P FORWARD DROP
[root@phoenix /]# iptables -t filter -P OUTPUT DROP
[root@phoenix /]# iptables -t nat -F
[root@phoenix /]# iptables -t nat -X
[root@phoenix /]# iptables -t nat -P PREROUTING ACCEPT
[root@phoenix /]# iptables -t nat -P OUTPUT ACCEPT
```

```
[root@phoenix /]# iptables -t nat -P POSTROUTING ACCEPT
[root@phoenix /]# iptables -t mangle -F
[root@phoenix /]# iptables -t mangle -X
[root@phoenix /]# iptables -t mangle -P PREROUTING ACCEPT
[root@phoenix /]# iptables -t mangle -P INPUT ACCEPT
[root@phoenix /]# iptables -t mangle -P FORWARD ACCEPT
[root@phoenix /]# iptables -t mangle -P OUTPUT ACCEPT
[root@phoenix /]# iptables -t mangle -P POSTROUTING ACCEPT
```

Vous remarquerez que l'on définie à "ACCEPT" les règles par défaut des tables "NAT" et "Mangle". Cela n'a pas trop d'influence sur la sécurité, du moment que vos règles de FORWARD sont bien écrites. Sinon, vous pouvez les mettre à "DROP", mais cela rallongera énormément le code et le temps de développement de votre script Netfilter.

Ce n'est pas non plus une mauvaise chose que de désactiver en début de script , au moins temporairement, le NAT (on ne sait jamais, des fois que vous l'ayez oublié) :

```
echo 0 > /proc/sys/net/ipv4/ip_forward
```

B. Chaînes utilisateurs

Nous avons peu parlé des chaînes utilisateurs, aussi nous allons y jeter un oeil. Lorsque vous écrivez vos règles Netfilter, il y a parfois des morceaux de code que vous aimeriez mettre en commun. Par exemple, supposons que vous voudriez interdire le "ping" de certaines machines du réseau local ainsi que du réseau externe, mais que vous vouliez aussi "logger" toute utilisation du "ping". Vous auriez alors à écrire quelque chose comme :

```
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.2 -p icmp -j LOG
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.2 -p icmp -j DROP
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.3 -p icmp -j LOG
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.3 -p icmp -j DROP
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -p icmp -j LOG
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -p icmp -j DROP
```

Dans ces cas, les règles utilisateurs sont là pour simplifier la vie. Commençons par écrire notre propre règle "LogDrop" qui comme son nom l'indique, va "logger" les trames, puis les supprimer :

```
[root@phoenix /]# iptables -t filter -N LogDrop
[root@phoenix /]# iptables -t filter -A LogDrop -j LOG --log-prefix LogDrop
[root@phoenix /]# iptables -t filter -A LogDrop -j DROP
```

Puis, appelons la pour nos pings sur les réseaux locaux :

```
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.2 -p icmp -j LogDrop
[root@phoenix /]# iptables -t filter -A INPUT -i eth0 -s 192.168.0.3 -p icmp -j LogDrop
```

Et les réseaux externes :

```
[root@phoenix /]# iptables -t filter -A INPUT -i eth1 -p icmp -j LogDrop
```

On note au passage que notre chaîne "LogDrop" peut être appelée à n'importe quelle occasion, avec une règle "parente" ayant ou non beaucoup d'options.

C. Script final d'exemple

Nous avons vu jusqu'à présent un bon nombre de fonctionnalités de Netfilter grâce aux commandes "iptables". J'ai donc écrit un script qui regroupe toutes ces fonctionnalités en un seul script que vous pouvez utiliser chez vous.

Avant toute chose, il faudra que vous le configuriez à votre usage. Pour cela, le début du script contient un certain nombre de variables globales, qui définissent le comportement du script :

LAN_* (Local Area Network) : ces variables définisent le réseau local. il s'agit du réseau " local.net".

WAN_* (Wan Area Network) : là, il s'agit des variables définissant le réseau Internet.

Elles sont spécialement conçues pour un réseau connecté par l'intermédiaire d'une passerelle. Or, vous devez plutôt avoir un modem RTC/RNIS/ADSL pour vous connecter, non ? Dans ce cas, il vous faudra donc décommenter le 2nd paquet de lignes "WAN_*", qui définirons vos paramètres de connexions Internet. La seule variable qu'il faudra bien vérifier est "WAN_INTERFACE=ppp0", mais à priori, vous ne vous connectez à Internet que par l'interface "ppp0". Dans le doute, et une fois que votre connexion Internet est activé, les commandes "/sbin/ifconfig" et "/sbin/route" vous donnerons plus d'informations.

NAT (Network Adress Translation) : cette variable définie si ou non vous voulez autorisez les machines de votre réseau interne à se connecter à Internet.

PF_* (Port Forwarding) : ces variables définissent si ou non vous voulez faire du port forwarding. Notez que j'ai pris ici l'exemple le plus simple :

le HTTP ne demande en effet qu'un seul port omnidirectionnel. Si vous voulez faire du port forwarding sur du FTP, ce sera un peu plus compliqué, car il faut laisser passer le canal de données (FTP-DATA) qui utilise le port 20 du coté client. Pour le P2P, c'est votre port de connexion (par exemple, "4660") qu'il faudra laisser passer en entrée.

Quelques remarques à propos de ce script :

Toutes les règles Netfilter qui contrôlent l'accès à Internet utilisent l'adresse IP externe de la machine ("WAN_IP"). C'est une manière de faire, qui je l'admets est contestable. Beaucoup de scripts que vous trouverez sur Internet n'utilisent pas ces options ("-d $WAN_IP" pour les règles "INPUT" et "-s $WAN_IP" pour les règles "OUTPUT"), et ne restreignent tout simplement pas les connexions externes à l'adresse IP externe. Par exemple, au lieu d'écrire :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o $WAN_INTERFACE -s $WAN_IP -d $WAN_NETWORK
\
-p all -m state --state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i $WAN_INTERFACE -s $WAN_NETWORK -d $WAN_IP
\
-p all -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Ces autres scripts écriraient :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o $WAN_INTERFACE -d $WAN_NETWORK \
-p all -m state --state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i $WAN_INTERFACE -s $WAN_NETWORK \
-p all -m state --state RELATED,ESTABLISHED -j ACCEPT
```

C'est effectivement plus simple à écrire, mais personnellement je n'ai pas confiance en ce type d'écriture "courte". Et ce, pour 2 raisons :

On peut toujours craindre une attaque malicieuse où un intrus nous enverrait un paquet mal formé, avec une adresse IP qui n'est pas la nôtre. C'est spécialement faisable si l'intrus se trouve être chez notre propre fournisseur d'accès. Ce paquet n'étant pas du tout normal, il n'a aucune raison de rentrer dans notre machine. On peut aussi imaginer qu'il indique comme adresse de destination une adresse de notre réseau interne. Ce type de paquet n'est pas du tout supposé pouvoir pénétrer dans notre machine, mais les techniques d'intrusion évoluant sans cesse, qui sait si un jour une telle technique ne sera pas capable de contourner les protections de Netfilter ?

"Iptables" permet d'être très "fin" au niveau de ses règles de filtrages, je trouve donc tout simplement regrettable de ne pas exploiter au maximum ses possibilités, et donc de ne pas faire des règles "ACCEPT" les plus restrictives possibles.

Conclusion : dans le doute, je m'abstiens, et j'utilise la "WAN_IP"...

Mais ce choix a un désavantage important : pour que ce script soit fonctionnel, il faut le lancer à chaque fois que la configuration réseau change, c'est à dire à chaque connexion à Internet. Et c'est la commande (barbare?) "**/sbin/ifconfig | grep "P-t-P" | sed "s/^[: a-z]*\([.0-9]*\).*/\1/g"**"

qui va se charger de trouver l'adresse IP internet de votre machine. Mais plutôt que de le lancer manuellement, vous pouvez laisser Linux s'en occuper pour vous. En effet, à chaque fois que la connexion ppp est initialisée, Linux exécute le script " /etc/sysconfig/network-scripts/ifup-ppp" (*).

Vous n'avez donc qu'à rajouter une ligne lançant ce script au début de ce fichier. Par exemple, quelque chose comme "/usr/local/sbin/iptables-final-1.sh".

Pour des raisons de sécurité, je vous conseille de donner la propriété de ce fichier au root, et de le laisser en écriture uniquement pour le super utilisateur.

Exemple :

```
[root@phoenix /]# chown root:root /usr/local/sbin/iptables-final-1.sh
[root@phoenix /]# chroot 755 /usr/local/sbin/iptables-final-1.sh
```

(*): en fait, ce n'est pas une obligation absolue. Et cela dépend en partie de l'outil que vous utilisez pour vous connecter sur Internet. Par exemple, pour ma connexion modem j'utilise l'interface graphique "/usr/bin/kppp". Et ce programme propose de lancer un script une fois la connexion Internet établie. C'est là que l'on pourrait mettre le " /usr/local/sbin/iptables-final-1.sh"

Dans ce script, vous pouvez voir que les règles de port forwarding sont situées avant celles d'IP masquerading. Simple effet de style de ma part ? Non, pas du tout. Un paquet venant d'Internet et à destination du port 80 de la machine satisfait à 2 règles de la chaîne "FORWARD" :

```
iptables -t filter -A FORWARD -i $WAN_INTERFACE -o $LAN_INTERFACE -s $WAN_NETWORK -d \
$LAN_NETWORK -p $PF_PROTO --dport $PF_PORT -m state --state ! INVALID -j ACCEPT
iptables -t filter -A FORWARD -i $WAN_INTERFACE -o $LAN_INTERFACE -s $WAN_NETWORK -d \
$LAN_NETWORK -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
```

De même que pour les paquets sortants de la machine et ayant pour source le port 80 :

```
iptables -t filter -A FORWARD -i $LAN_INTERFACE -o $WAN_INTERFACE -s $LAN_NETWORK -d \
$WAN_NETWORK -p $PF_PROTO --sport $PF_PORT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -i $LAN_INTERFACE -o $WAN_INTERFACE -s $LAN_NETWORK -d \
$WAN_NETWORK -p all -m state --state ! INVALID -j ACCEPT
```

Dans les 2 cas, la 1ère règle est une règle de port forwarding et la 2nd d'IP masquerading. Mais comme les 2nd règles sont plus générales que les premières, les statistiques de Netfilter (commande "iptables -L -n -v -t filter") sont faussées si les règles d'IP masquerading sont placées avant les règles de port forwarding.

Cet exemple est intéressant, car c'est un des fameux cas où l'ordre des commandes "iptables" a une certaine importance. Comment cela je chipote ? Et bien oui...

Notez enfin que les différents modules de Netfilter ("iptable_nat", "ip_nat_ftp", etc ...) ne sont chargés que si c'est nécessaire. On aurait par contre pu tous les charger "en bloc" au début du script.

D. Tests d'intrusion

Une fois que toutes vos règles iptables sont écrites, il vous faut vous-même tester la sécurité de votre Linux en pratiquant des tests d'intrusions. Une première approche est d'utiliser "nmap", et de contrôler chacun de vos ports. Cette pratique est très efficace (lisez "man nmap" afin d'utiliser au mieux les options très variées de Nmap), et doit être fait depuis les machines de votre réseau. Pendant vos "attaques" lancées par "nmap", n'oubliez pas de garder un oeil sur votre "/var/log/messages" (par exemple avec un "tail -f /var/log/messages") ou le fichier qui stocke les logs de Netfilter, afin de suivre la progression de l'attaque.

Dans le cadre du réseau proposé ici, il m'a été facile de tester les 2 interfaces réseaux de Phoenix, en lançant les tests depuis Paradise et Pirate.

Cependant, dans le cadre d'une utilisation personnelle, il est beaucoup plus difficile de tester l'interface réseau externe ("ppp0" par exemple). En effet, si par exemple depuis Phoenix vous lancez un "nmap phoenix1.internet.net", vous ne ferez que tester les règles de "loopback/localhost", comme nous l'avons déjà vu précédemment . Pour que vous obteniez un résultat correct, il vous faut donc changer vos règles de "loopback" par celles que vous avez définies pour "ppp0". Mais cette solution n'est pas forcément très représentative de la réalité, et le mieux serait de tester réellement les paquets arrivant sur "ppp0".

Il vous faut donc faire confiance à une personne externe à votre réseau, situé sur Internet, pour tester vos règles de Firewall. Pour cela, 3 possibilités :

Vous avez un accès en "telnet" ou en "SSH" (ce qui est mieux, car plus sécurisé) sur une machine situé sur Internet. Vous vous connectez alors dessus, et vous lancez un "nmap" sur l'adresse IP de votre machine. Attention de ne pas vous tromper d'adresse IP, sinon vous iriez tester les ports d'une autre machine, et son propriétaire pourrait ne pas être très content...

Vous demandez à un de vos amis de faire ce test pour vous. Encore faut il qu'il sache ce qu'est "nmap" !

Enfin, vous faire faire ce test par un site web. Certains sites proposent gratuitement ce genre de prestation, entre autre pour vous proposer des solutions de firewall ... Windows®

En fait, tout le mérite en revient au suivi de connexion, qui fait passer notre machine pour un "trou noir". Comme le veut la définition de ce terme, c'est "un corps qui absorbe tout, et qui ne rejette rien"...

Quelques soit ces sites, leurs tests ne sont souvent pas très poussés, et se limitent parfois qu'aux seuls ports ouverts par défaut sous Windows®. De plus, un simple test de "port scanning" n'est pas forcément suffisant, et d'autres outils de tests d'intrusion peuvent être utilisés en complément. Mais à ce niveau là, la limite entre tests de sécurité et piratage est très tenue, donc je vous laisserai chercher par vous-même des moyens de tester un peu plus en profondeur votre système... Quelques soit ces sites, leurs tests ne sont souvent pas très poussées, et se limitent parfois qu'aux seuls ports ouverts par défaut sous Windows®. De plus, un simple test de "port scanning" n'est pas forcément suffisant, et d'autres outils de tests d'intrusion peuvent être utilisés en complément. Mais à ce niveau là, la limite entre tests de sécurité et piratage est très tenue, donc je vous laisserai chercher par vous-même des moyens de tester un peu plus en profondeur votre système...

E. Sauvegarde des règles Netfilter

Supposons que vous n'écriviez pas votre adresse IP Internet dans vos règles Netfilter de "ppp0" (comme décrit ci-dessus), vous avez alors des règles de ce type :

```
[root@phoenix /]# iptables -t filter -A OUTPUT -o ppp0 -d 0.0.0.0/0 -p all -m state \
--state ! INVALID -j ACCEPT
[root@phoenix /]# iptables -t filter -A INPUT -i ppp0 -s 0.0.0.0/0 -p all -m state \
--state RELATED,ESTABLISHED -j ACCEPT
```

Dans ce cas, vos règles Netfilter sont génériques , et elles n'ont pas besoin d'être modifiées à chaque connexion Internet. Donc vous pouvez écrire vos règles une fois pour toute, et laisser Linux les charger automatiquement au démarrage de votre machine. Pour cela, écrivez vos règles Netfilter dans un script, puis (pour les possesseurs de distributions Mandrake) tapez la commande :

```
[root@phoenix /]# /etc/rc.d/init.d/iptables save
```

Enfin, pour que le Netfilter soit opérationnel à chaque démarrage, il faut créer les liens symboliques adéquates dans le /etc/rc.d/ :

```
[root@phoenix /]# cd /etc/rc.d/rc3.d/
[root@phoenix /]# ln -s ../init.d/iptables s03iptables
[root@phoenix /]# cd /etc/rc.d/rc5.d/
[root@phoenix /]# ln -s ../init.d/iptables s03iptables
```

Au démarrage, votre Linux lancera le script /etc/rc.d/init.d/iptables qui chargera les règles Netfilter précédemment sauvées. Si à un moment vous avez besoin d'arrêter la protection de Netfilter, vous n'avez qu'à lancer la commande suivante :

```
[root@phoenix /]# /etc/rc.d/init.d/iptables stop
```

VIII. Firewall applicatif

Jusqu'à présent, les utilisateurs de Windows® ont pu comparer le fonctionnent des techniques de Firewall de Linux avec ce qu'ils trouvent sur leur OS.

Et bien que les fonctionnalités de Netfilter sont au moins équivalentes à ce que l'on trouve dans les outils de firewall sous Windows®, il reste une fonctionnalité qui manque à Linux : c'est le firewall applicatif.

On appelle "firewall applicatif" la capacité d'un logiciel à contrôler à la fois les flux de données entrantes et sortantes en fonction de certains critères (adresse IP, port, type de connexion, etc...), mais aussi des applications qui exploitent ces connexions. En gros, un firewall applicatif va permettre de contrôler quelles sont les connexions réseau que fait tel ou tel logiciel .

C'est typiquement ce que font des applications fonctionnant sous Windows®, tel que par exemple ZoneAlarm®, TinyFirewall®, Lock 'n' Stop®, etc... A chaque connexion que tente une application, ces firewall applicatifs vérifient que la connexion est autorisée, ou alerte l'utilisateur.

Mais à quoi sert exactement ce type d'application ? En fait, elles permettent à l'utilisateur de surveiller si telle ou telle application tente une connexion non autorisée , quasiment dans le dos de l'utilisateur. Par exemple lorsque Word® ou Excel® font une connexion Internet sans raison apparente, que le MediaPlayer® envoie une requête alors qu'il est en pleine lecture de vidéo, ou qu'une dll/ActiveX/autre utilise Internet Explorer® pour accéder à Internet.

L'utilisateur de Windows n'aurait donc pas confiance aux applications qu'il utilise ? Oui, c'est du moins ce qui ressort des discussions des utilisateurs de Windows et de firewall applicatifs. Et pourquoi ne passent ils pas sous Linux alors ? Parce que ce type de fonctionnalité n'existe pas peut-être ? Que nenni, c'est faisable aussi sous Linux!

La mise en place d'un système de firewall applicatif se fait en fait en 2 temps :

Premièrement, il faut charger le module du kernel appelé "ip_queue"

```
[root@phoenix /]# modprobe ip_queue
```

Puis dans les règles de Netfilter, par exemple au niveau des chaînes INPUT et OUTPUT de la table "Filter", il faut créer une règle qui envoie tout les paquets suspects vers la cible "QUEUE" (nous ne l'avons pas encore vue jusqu'ici). L'idéal est de la mettre en tant que dernière règle de la chaîne, afin que les autres règles aient déjà filtrés les paquets :

```
[root@phoenix /]# iptable -t filter -A OUTPUT -j QUEUE
[root@phoenix /]# iptable -t filter -A INPUT -j QUEUE
```

Dans le jargon du kernel, on dit que l'on envoie le paquet dans le "user space" ("espace utilisateur" en français)

Enfin, il faut avoir un applicatif qui va recevoir ces paquets, les analyser en fonction par exemple du programme qui a fait la connexion ou à qui elle est destinée, et refusé ou non le paquet. Une interface graphique peut par exemple montrer à l'utilisateur ce qui se passe, libre à ce dernier d'accepter ou non le paquet.

Pour l'instant, il n'existe que peu d'applications sous Linux qui fournissent de telles fonctionnalités "toutes faites". L'un d'entre eux est FireFlier., ce logiciel gère à la fois les règles Netfilter classiques, ainsi que les règles spécifiques aux applications.

Vous pouvez évidement développer votre propre firewall applicatif. Pour cela, vous aurez besoin :

- de la **LIBIPQ**,
- éventuellement de **ipqmpd** qui permet de récolter les paquets IP venant de Netfilter
- de quelques neurones, afin de créer votre programme de gestion des paquets, et des règles d'entrée / sorties.